Bachelorarbeit

Middleware-Plattform zur Harmonisierung von Smart Home- und IoT-Systemen

Studiengang Informatik
OST – Ostschweizer Fachhochschule
Campus Rapperswil-Jona

Frühjahrssemester 2024

Autor	Marc Kissling
Betreuer	Prof. Dr. Olaf Zimmermann
Projektpartner	Cloud Application Lab (CAL) des Instituts für Software (IFS)
Experte	Dr. Gerald Reif
Gegenleser	Nikolaus Heners

Anerkennung und Dankbarkeit

Ein herzliches Dankeschön geht an die Fachhochschule Ost am Standort Rapperswil für die finanzielle Unterstützung, insbesondere für die Deckung der Cloud-Kosten, die durch die SmartBridge angefallen sind.

Zudem möchte ich mich bei Prof. Dr. Olaf Zimmermann für seine fachkundige Anleitung und Unterstützung während des gesamten Prozesses bedanken.

Ebenso danke ich Dr. Gerald Reif für sein wertvolles Feedback als Experte und Nikolaus Heners für seine Einschätzungen als Gegenleser.

Ihre Beiträge haben massgeblich dazu beigetragen, diese Arbeit zu verbessern und zu einem erfolgreichen Abschluss zu bringen.

Gender-inklusiver Hinweis

Bei der Verwendung von Begriffen, die auf ein bestimmtes Geschlecht hinweisen, werden alle Geschlechter und Identitäten gleichermassen anerkannt und respektiert. Wir akzeptieren Menschen unabhängig von ihrem Geschlecht, ihrer Identität oder ihrem Ausdruck.

Abstract

Einleitung und Problemstellung
Die zunehmende Digitalisierung und Vernetzung von Haushaltsgeräten hat zu einer Vielzahl proprietärer Smart Home- und IoT-Systemen von verschiedenen Herstellern geführt. Diese Vielfalt stellt erhebliche Herausforderungen für Nutzer dar, die ein nahtloses und einheitliches Smart Home Erlebnis anstreben. Nutzer müssen oft verschiedene Apps und Plattformen verwenden, um ihre Geräte zu steuern, was zu einer fragmentierten und ineffizienten Handhabung führt. Gleichzeitig stehen Entwickler von Geräteherstellern vor der Schwierigkeit, universelle Schnittstellen zu erstellen, die mit den unterschiedlichen Kommunikationsprotokollen und Datenformaten der proprietären Systeme kompatibel sind. Inkompatibilitäten behindern die Schaffung einer kohärenten Benutzererfahrung und erschweren die Integration neuer Geräte und Funktionen.

Ziel der Arbeit
Die Bachelorarbeit zielt darauf ab, eine Middleware-Plattform namens "Smart-Bridge" zu entwickeln. Diese Plattform soll die Integration und Harmonisierung verschiedener Smart Home- und IoT-Systeme erleichtern, indem sie eine einheitliche Schnittstelle bereitstellt. Diese Schnittstelle ermöglicht es Frontend-Entwicklern, benutzerfreundliche Visualisierungen und Steuerungen zu erstellen, ohne sich mit den technischen Details der verschiedenen Systeme auseinandersetzen zu müssen. Darüber hinaus erlaubt die Smart-Bridge Geräteherstellern, sich auf Innovationen zu konzentrieren. Da die Hersteller unterschiedliche Herangehensweisen verfolgen, ist eine eigene Abstraktionsschicht wichtig.

Fazit
Die konzipierte Middleware-Plattform "SmartBridge" löst erfolgreich die Herausforderungen bei der Integration verschiedener Smart Home- und IoT-Systeme (Sonos-, Shelly- und KNX-Umgebung). Sie bietet eine skalierbare, flexible und benutzerfreundliche Lösung, die die Interoperabilität zwischen verschiedenen Geräten und Systemen verbessert. Ein Proof of Concept bestätigt die technische Machbarkeit und zeigt auf, dass eine systemunabhängige Middleware die Nutzererfahrung im Smart Home-Bereich verbessern kann. Wichtig ist, bei der Architektur die Module nicht zu feingranular zu gestalten, um die Effizienz zu steigern.

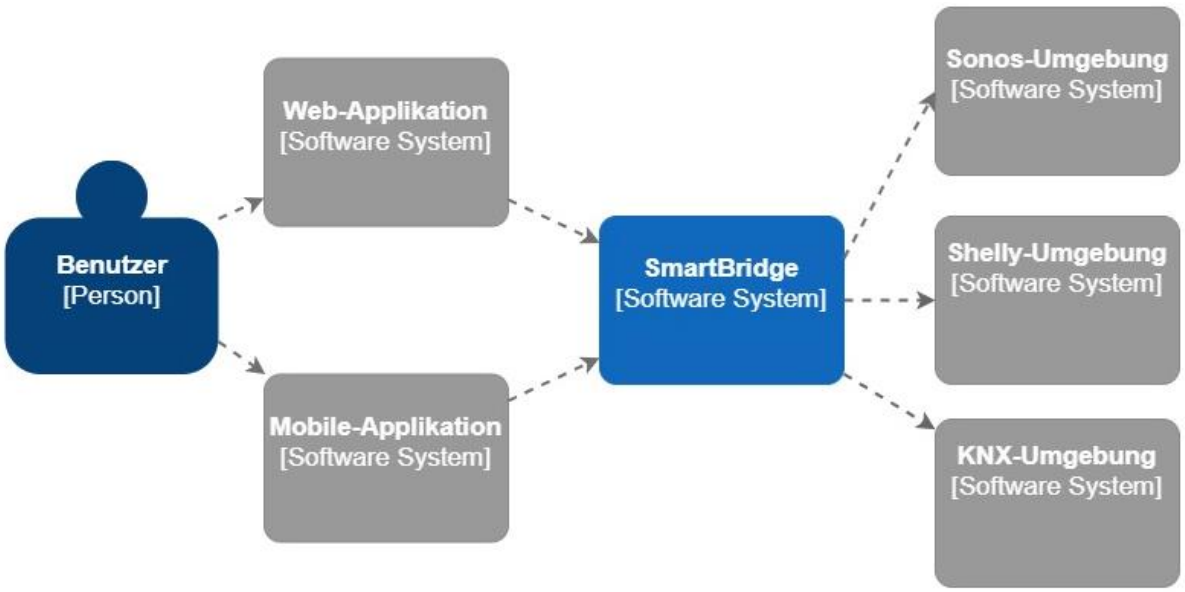


Abbildung 1: Übersichtsdiagramm der Middleware

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung des Dokumentes

Diese Kapitel richtet sich sowohl an die technisch interessierten Endbenutzer als auch an die Hersteller der SmartHome- und IoT-Systeme und die User-Interface-Entwickler.

1.1. Ausgangslage und Problemstellung

Können SmartHome und IoT-Geräte verschiedener Hersteller miteinander interagieren und auf einem einzigen User-Interface bedient werden?

Mit der zunehmenden Digitalisierung der Haushalte und der verschiedenen Geräte, die sie beherbergen, kommen immer mehr Apps verschiedener Hersteller auf den Markt. Die Produzenten sehen sich verpflichtet ausserhalb ihres Kernbereiches, der Geräteentwicklung, auch User Interfaces wie Mobile-Apps oder Web-Portale zu entwickeln. Zudem sind viele der Geräte nicht mit den Produkten oder User-Interfaces anderer Hersteller kompatibel.

Dies führt dazu, dass viele Haushalte unzählige User Interfaces auf Ihren Handys, Tablets und PC's haben, jedoch durch die Anzahl der verschiedenen Interfaces schnell die Übersicht verliert, welches Gerät von welchem Interface gesteuert und verwaltet wird. Genau da setzt diese Arbeit an.

1.2. Zielsetzung

Die Idee ist es für die gängigsten SmartHome und IoT-Geräte eine Middleware zu entwickeln, an welche die Geräte-Hersteller einen Konnektor erstellen und veröffentlichen können. Zudem steht User-Interface-Entwicklern eine einheitliche RESTful HTTP Web API zur Verfügung. So können diese eigene User-Interfaces erstellen und der Endbenutzer kann sich für das für ihn passende Interface entscheiden.

Die Middleware wird als Cloud-Middleware betrieben, was den Endbenutzern so die Möglichkeit bietet von überall her mit einer Internetverbindung ihr SmartHome und ihre IoT-Geräte zu steuern.

Die Arbeit entspricht dem Charakter eines Proof-of-Concepts und soll prüfen ob es möglich ist, eine fremd-system-unabhängige Middleware zur Vereinheitlichung der User-Interaktion zu implementieren.

Im Rahmen dieser Arbeit wurden Konnektoren für die Hersteller Sonos und Shelly sowie für den Gebäudeautomationsstandard KNX erstellt.

1.3. Erläuterung der Begriffe

Zur besseren Verständlichkeit werden nachfolgend einige Fachbegriffe kurz erläutert:

Konnektor
Unter dem Begriff Konnektor ist in diesem Dokument eine eigenständige Applikation zu verstehen, welche die Zwischenschicht zwischen dem Fremd-System und der Smart-Bridge-Middleware darstellt. Zu ihren Aufgaben gehört das Austauschen von Nachrichten und damit verbunden das Anreichern und Interpretieren von fehlenden Informationen.

Middleware
Die Smart-Bridge-Middleware ist eines der Arbeitsziele dieser Arbeit. Es handelt sich dabei um eine cloudbasierte Software, welche verschiedenste Funktionalitäten bereitstellt. Die Implementation der Middleware ist hierbei unabhängig von den Fremd-Systemen und den User-Interface-Implementierungen. In Richtung der Fremd-Systeme wird über eine Messaging-Plattform mit den Konnektoren kommuniziert. Den User-Interface-Implementierungen wird eine 13.2.1.40 RESTful HTTP Web API zur Verfügung gestellt. *Siehe dazu auch Kapitel 13.2.1.40 RESTful HTTP Web API.*

Endbenutzer
Mit dem Ausdruck Endbenutzer sind in dieser Arbeit alle Anwender der Middleware, Konnektoren und User-Interfaces vereint.

User-Interface
Der Begriff User-Interface inkludiert im Rahmen dieser Arbeit alle Eingabesoftware (Web-Applikation & Mobile-Applikation), die einem Endbenutzer zur Verfügung gestellt wird, um eine Art von Interaktion mit den Fremd-System-Komponenten zu vollziehen. Hierbei ist zu erwähnen, dass hierfür die API der Middleware genutzt werden muss.

Repräsentation
Eine Repräsentation meint im Rahmen dieser Arbeit eine teilweise Abbildung eines Objektes, über welche für den spezifischen Anwendungsfall alle relevanten Informationen vorhanden sind. Dies kann jedoch auch nur eine Teilmenge aller Informationen des abgebildeten Objektes sein. Zudem können auf der Repräsentation zusätzliche Hintergrundinformationen enthalten sein.

Fremd-System
Ein Fremd-System bezeichnet ein Eco-System eines Herstellers von physischen Geräten und / oder Software im Umfeld von IoT und Smart Home. Hierbei spielt es keine Rolle, ob die System-Komponenten direkt oder via Server des Herstellers erreichbar sind. Beispiele diesbezüglich sind Sonos, KNX und Shelly.

Sensor
Ein Sensor ist ein Gerät, welches einen Zustand registriert und diesen bei einer Änderung anderen Komponenten mitteilt. Die Zustandsänderung kann bei einem Temperatur-Sensor die Temperatur-schwankung oder bei einem Strohm-Schalter, dass schliessen eines Stromkreises sein.

Aktor
Ein Aktor ist ein Gerät, welches eine Aktion auslöst. Dies kann im Falle eines Schalt-Aktors das Schliessen eines Stromkreises sein, was damit das Einschalten einer Lampe bewirkt.

1.4. Aufgabenstellung

Können SmartHome und IoT-Geräte verschiedener Hersteller miteinander interagieren und auf einem einzigen User-Interface bedient werden?

Mit der zunehmenden Digitalisierung der Haushalte und der Vernetzung verschiedener Geräte, kommen immer mehr Apps zahlreicher Hersteller auf den Markt. Die Produzenten sehen sich verpflichtet ausserhalb ihres Kernbereiches, der Geräteentwicklung, auch User Interfaces wie Mobile-Apps oder Web-Portale zu entwickeln. Dabei sind viele der Geräte nicht mit den Produkten oder User-Interfaces anderer Hersteller kompatibel.

Dies führt dazu, dass viele Haushalte unzählige User-Interfaces auf Ihren Handys, Tablets und PC's haben, jedoch durch die Anzahl der verschiedenen Interfaces schnell die Übersicht verlieren, welches Gerät von welchem Interface gesteuert und verwaltet wird. Genau bei diesem Umstand setzt diese Arbeit an.

1.4.1. Ausgangslage

Zahlreiche Hersteller bieten Produkte und Plattformen in den Bereichen Smart Home, Anlagensteuerung und Internet of Things (IoT) an. In vielen Systemen in diesen Bereichen zeigt sich eine Tendenz zur Isolation der Produktpalette; ihre Nutzer bewegen sich typischerweise in den spezifischen Ökosysteme ihrer Hersteller. Die proprietären Systeme sind nicht kompatibel mit Produkten anderer Hersteller und Kommunikationsprotokolle sind nicht interoperabel; die Integration und Kommunikation zwischen Geräten unterschiedlicher Anbieter ist daher schwierig. Wer ein vernetztes System für Zuhause, für industrielle Anlagen oder für andere IoT-Anwendungen implementieren möchte, muss entweder innerhalb der Grenzen eines Hersteller-Ökosystems bleiben oder sich für offene, aber möglicherweise kostspielige und komplexe Lösungen entscheiden. Entwicklerteams agieren oft aus der Perspektive ihres eigenen Unternehmens und können kein breiteres Verständnis für die Bedürfnisse und Erwartungen der Endnutzer entwickeln. Die Strukturen und Schnittstellen der Systeme werden nur aus einer technischen Sicht entworfen. Dies führt zu Herausforderungen für Endbenutzer und für Hersteller von Visualisierungs- und Steuerungssoftware; die Schaffung von kohärenten, gut benutz- und wartbaren Systemen ist aufwändig und fehlerträchtig.

1.4.2. Ziele der Arbeit und Liefergegenstände

Im Rahmen der Bachelorarbeit soll eine Integrations-Plattform (Middleware) konzipiert und entwickelt werden, die es User-Interface Entwicklern ermöglicht, benutzerfreundliche Visualisierungen und Steuerungen zu kreieren, ohne sich in technische Protokolldetails vertiefen zu müssen. Parallel dazu können sich Hardwarehersteller auf ihre Kernkompetenzen konzentrieren und innovative Geräte entwickeln, ohne den Druck der User-Interface Entwicklung zu spüren.

1.4.3. Zielgruppe der Leser

Diese Arbeit soll auch als Inspiration und Knowhow-Quelle für ähnliche Arbeiten dienen, vorwiegend für technikinteressierte Personen. Da ich mir jedoch bewusst bin, dass die Tragweite dieser Arbeit kaum über die DACH-Region hinaus gehen wird, habe ich mich aktiv dafür entschieden die Arbeit in deutscher Sprache zu verfassen, um die Zugänglichkeit zu erhöhen.

Sowohl Hersteller von IoT- und Smart Home-Geräten als auch Mobile- und Web-Entwickler mit Interesse an IoT- und Smart Home sollen angesprochen werden. Die Middleware soll als Schnittstelle fungieren, um beide Parteien zu vereinen und ihre jeweiligen Stärken zu nutzen. Hersteller sollen die Verbindung zwischen ihren Produkten und der Middleware herstellen, während die User-Interface-

Entwickler die Schnittstelle zwischen der Middleware und ihren User-Interface-Produkten bilden. Ziel ist es, dem Endbenutzer eine möglichst positive Benutzererfahrung zu bieten.

1.5. Stakeholder

Da für jede Stakeholder-Gruppe unterschiedliche Informationen von Interesse sind wurde folgende Aufstellung erstellt welche abbildet, welche Kapitel für welche Stakeholdergruppe interessant sein könnten:

Kapitel	**Stakeholdergruppen**		
	1.5.1 - Endbenutzer (S. 15)	1.5.2 - Hersteller von IoT- und Smart Home-Geräten (S. 16)	1.5.3 - User-Interface-(Software)Entwickler (S. 17)
1 - Einleitung des Dokumentes (S. 11)	x	x	x
2 - Randbedingungen (S. 20)	-	x	(x)
3 - Kontextabgrenzung (S. 22)	x	x	x
4 - Lösungsstrategie (S. 37)	-	x	x
5 - Building Block View (S. 53)	-	x	x
6 - Laufzeitsicht (S. 94)	x	x	x
7 - Verteilungssicht (S. 101)	-	(x)	x
8 - Querschnitt-Konzepte (S. 103)	-	x	x
9 - Design-Entscheidungen (S. 110)	(x)	x	x
10 - Qualitätsanforderungen (S. 115)	(x)	x	x
11 - Risiken und Erkenntnisse (S. 123)	-	x	x
13 - Glossar (S. 130)	x	-	-

Tabelle 1: Kapitelübersicht der Stakeholdergruppen

Legende:

X	Bringt der Stakeholdergruppe einen Mehrwert
(x)	Bringt der Stakeholdergruppe keinen direkten Mehrwert
-	Kann interessant für die Stakeholdergruppe sein

Unter Berücksichtigung des allgemeinen Verständnisses jeder Stakeholder-Gruppe wurde darauf geachtet, dass einzelne Text-Passagen übersprungen werden können, ohne dass inhaltliche Informationen verloren gehen.

Als Zielgruppen wurden drei Stakeholder-Gruppen, welche jeweils einen unterschiedlichen Hintergrund und damit hergehend einen unterschiedlichen Wissensstand in den jeweiligen Themen besitzen, eruiert. Es sind dies die Endbenutzer, welche die SmartHome und IoT-Apps anwenden, die Produkte-Hersteller von IoT und SmartHome-Geräten, welche einen Konnektor für den eigenen proprietären Standard implementieren und die User-Interface Entwickler, welche eine Schnittstelle zur Entwicklung der Applikation verwenden.

Um die typischen Merkmale und Verhaltensweisen der einzelnen Zielgruppen besser zu verstehen, werden nachstehend die Personas zu den einzelnen Zielgruppen erläutert.

1.5.1. Endbenutzer

Persona Einfamilienhaus-Besitzer
Der 45–50-jährige Einfamilienhaus-Besitzer, welcher nach seiner Technischen Lehre zum Werkzeugmacher und einer betriebswirtschaftlichen Weiterbildung im Management eines Schweizer KMU's beschäftigt ist. Er ist glücklich verheiratet und hat zwei Kindern. Sowohl im beruflichen als auch im privaten Umfeld legt er grossen Wert auf Genauigkeit, Sauberkeit und Pünktlichkeit.

Da sein Arbeitsalltag geprägt ist von Managementaufgaben geht er in seiner Freizeit seinem technischen Flair nach und beschäftigt sich gerne mit den neuesten technischen Entwicklungen, die für die breite Öffentlichkeit erschwinglich ist.

Wünsche
Der Endbenutzer möchte ein zuverlässiges Smart Home- und IoT-System, das stabil und sicher funktioniert. Doch möchte er sich nicht aktiv um dessen Betrieb kümmern müssen. Das Einrichten soll schnell und einfach von der Hand gehen. Dabei möchte er vorzugsweise nicht auf einen Fachmann zurückgreifen müssen.

Bedürfnisse
Die Bedürfnisse des Endbenutzers sind die folgenden:

- *Automatisierung*
  Der Smart Home Assistent bietet umfassende Automatisierungsfunktionen, die es dem Hausbesitzer ermöglichen, seine Haustechnik zu optimieren. Beispielsweise kann das System automatisch das Licht einschalten, wenn jemand den Raum betritt.
- *Sicherheit*
  Das System basiert auf erprobten Sicherheitsmechanismen, welche dem Hausbesitzer ein beruhigendes Gefühl vermitteln. So muss sich der Hausbesitzer keine Gedanken über unberechtigte Zugriffe auf den Smart Home Assistenten machen.
- *Benutzerfreundlichkeit*
  Die Einrichtung des Smart Home Assistant funktioniert schnell und einfach, ohne dass ein Fachmann hinzugezogen werden muss. Der Hausbesitzer kann das System einfach konfigurieren, Einstellungen anpassen und auf alle Funktionen zugreifen.
- *Kompatibilität*
  Der Smart Home Assistent ist mit einer Vielzahl von SmartHome-Geräten und -Plattformen kompatibel, so dass der Hausbesitzer die Flexibilität hat, sein Smart Home-System nach seinen eigenen Vorlieben zu gestalten und zu erweitern.

1.5.2. Hersteller von IoT- und Smart Home-Geräten

Persona Inhaber und Geschäftsführer
Der Inhaber und Geschäftsführer eines Familienunternehmens, das inzwischen bereits in der dritten Generation geführt wird, hat den Betrieb von einem Handelsunternehmen für Elektronik-Bauteile zu einem Elektronikkomponenten-Hersteller entwickelt. Das Unternehmen wird weiterhin als KMU eingestuft und verfügt nicht über die Ressourcen und Marktmacht grosser Konzerne.

Um sich den aktuellen Markt-Anforderungen anzupassen und auch SmartHome- und IoT-kompatible Produkte anzubieten, ohne jedoch grosse finanzielle Aufwendungen zu haben, entscheidet sich der Inhaber und Geschäftsführer, ein eigenes proprietäres System auf den Markt zu bringen. Die Entwicklung soll aus Kostengründen mit den vorhandenen Mitarbeitern geschehen.

Um dem Kunden die Möglichkeit zu geben auch mit anderen Smart Home- und IoT-Systemen kompatibel zu sein, ohne an andere Systeme Verbindungen zu implementieren, beschliesst der Inhaber zusammen mit dem Produkt-Managementteam, dass ein Mitarbeiter (35-Jährig, Lehre als Systemtechniker) in den nächsten Wochen einen Konnektor für den eigenen proprietären Standard an die SmartBridge-Middleware implementieren soll.

Wünsche
Eine einfach verständliche Beschreibung darüber, welche Informationen übermittelt werden und wie das Schema aussehen sollte. Zudem wird eine Unabhängigkeit von Programmiersprachen angestrebt.

Bedürfnisse
Die Bedürfnisse des Herstellers von IoT- und Smart Home-Geräten wurden wie nachstehend beschrieben identifiziert:

- Einfache, verständliche Beschreibung:
  Der Konnektor sollte eine leicht verständliche Beschreibung enthalten, welche Informationen übermittelt werden müssen, um eine reibungslose Integration zu ermöglichen. Dies umfasst Informationen wie Geräteidentifikation, unterstützte Funktionen und das Kommunikationsprotokoll.

- Übersichtliches Schema:
  Das Schema des Konnektors sollte klar strukturiert sein und alle erforderlichen Felder enthalten, um die Interaktion mit anderen Smart Home-Plattformen zu erleichtern. Dies umfasst Felder für Geräteinformationen, Steuerbefehle und Ereignismeldungen.

- Programmiersprachen-Unabhängigkeit:
  Der Konnektor sollte programmiersprachenunabhängig sein um sicherzustellen, dass er in verschiedenen Umgebungen und Technologie-Stacks eingesetzt werden kann. Dies ermöglicht eine breitere Akzeptanz und Nutzung des Konnektors.

- Vollständige Dokumentation:
  Eine umfassende Dokumentation des Konnektors soll erstellt werden, welche alle technischen Details sowie Anweisungen zur Konfiguration, Installation und Verwendung enthält. Dies erleichtert es anderen Entwicklern im Unternehmen, den Konnektor zu verstehen und effektiv damit zu arbeiten.

1.5.3. User-Interface-(Software)Entwickler

Persona User-Interface-Entwickler

Vor knapp 20 Jahren hat der User-Interface-Entwickler eine Lehre zum Elektroniker erfolgreich abgeschlossen. Mit der zunehmenden Verbreitung des PC's und des Internets hat dieser erst in den Informatik-Support und später in die Software-Entwicklung gewechselt. Mit kontinuierlichen Weiterbildungen hat er sich einen grossen und breiten Erfahrungsschatz im Erstellen von Webseiten mit JavaScript angeeignet.

Da er seine eigene Vorstellung hat, wie eine Benutzeroberfläche eines SmartHome- oder IoT-Systems aussehen soll, implementiert er in seiner Freizeit ein solches System. Da er seine Leidenschaft gerne mit andern teilt, entwickelt er die Benutzeroberflächen in einem Open-Source-Projekt mit ca. 15 Gleichgesinnten.

Wünsche

Der User-Interface Entwickler, welcher ein User-Interface für die Middleware entwickelt, möchte eine API zur Verfügung haben. Die Dokumentation soll in einem ihm bekannten Standard geschrieben sein und der Austausch der Daten soll für ihn nichts grundlegend Neues darstellen.

Bedürfnisse

Die Bedürfnisse des User-Interface Entwicklers unterscheiden sich von jenen des Endbenutzers oder Geräteherstellers und können wir folgt beschrieben werden:

- *Gut dokumentierte API*
  Der User-Interface Entwickler benötigt Zugriff auf eine gut dokumentierte API, um das User Interface für die Middleware zu entwickeln. Die API-Dokumentation soll alle verfügbaren Endpunkte, Parameter, Rückgabewerte und Authentifizierungsmethoden klar und präzise beschreiben.

- *Bekannter Standard*
  Die API-Dokumentation soll in einem Standardformat verfasst sein, welches der Entwickler bereits kennt und mit welchem er vertraut ist.

- *Einfacher Datenaustausch*
  Der Austausch von Daten zwischen dem User-Interface und der Middleware soll für den Entwickler nichts grundlegend Neues darstellen. Die Middleware soll eine klare und konsistente Datenstruktur verwenden die leicht zu verstehen und zu verarbeiten ist. Dies ermöglicht es dem Entwickler effizient und ohne grössere Schwierigkeiten mit den bereitgestellten Daten zu arbeiten.

- *Hohe Flexibilität*
  Die API soll dem Entwickler eine gewisse Flexibilität bieten, um die Benutzeroberfläche nach seinen eigenen Vorstellungen zu gestalten. Dies umfasst die Möglichkeit, benutzerdefinierte Widgets, Layouts und Interaktionsmuster zu implementieren, um eine optimale Benutzererfahrung zu gewährleisten.

1.6. Qualitätsziele

Nachdem die Bedürfnisse der Zielgruppen bekannt sind, werden die entsprechenden Qualitätsziele formuliert. Diese werden nach allgemeinen und komponenten-spezifischen Qualitätszielen unterteilt.

Da der branchenübliche Standard ISO/IEC 25010:2023 sehr ausführlich dargestellt ist, wurden sowohl das FURBS+ als auch das Q42 Qualitätsmodell in Betracht gezogen. Da das Q42 Qualitätsmodell von denselben Personen stammt, die auch die Dokumentations-Vorlage arc42erstellt haben (welche eine Vorlage dieses Dokuments ist), kommt das Q42 Modell zur Anwendung.

Allgemeine Qualitätsziele
Über das für den Endbenutze zur Verfügung gestellte Gesamtsystem (Konnektoren, Middleware und Demo-User-Interface) wurden die nachstehenden Qualitäts-Attribute definiert. Diese werden mit zunehmender Granularität der Teil-Systeme weiter ausgeführt.

- Zuverlässigkeit
- Nutzbarkeit
- Flexibilität
- Operabilität

Komponenten-spezifische Qualitätsziele
Zusätzlich zu den allgemein Qualitätszielen sind folgende Qualitätsziele spezifisch für Teilbereiche von Relevanz:

- Messaging effizient (efficient)
- Smart-Bridge-Core sicher (save)
- Smart-Bridge-Core geeignet (suitable)

Dieser Abschnitt behandelt nur die Allgemeine Qualitätsziele. Die Komponenten-spezifische Qualitätsziele werden im Kapitel 10 - Qualitätsanforderungen behandelt.

1.6.1. Allgemeine Qualitätsziele

Dieser Unterabschnitt behandelt die an alle Bereichen des Ecosystems gestellten Anforderungen in Bezug auf dessen Qualität.

1.6.1.1. Zuverlässigkeit (reliable)

Spezifikation nach arc42 Quality Model:
„Perform specified functions under specified conditions without interruptions or failures.“
(Starke, arc42 Quality Model, 2022)

Endbenutzer müssen sich jederzeit auf die Funktionstüchtigkeit des Gesamt-Systems verlassen können. Ein Endbenutzer muss darauf vertrauen können, dass die Fenster tatsächlich geschlossen sind, wenn er sie über das System schliesst, und dass sie nicht während seiner Abwesenheit offen bleiben oder sich aufgrund eines Fehlers plötzlich öffnen.

1.6.1.2. Brauchbarkeit (usable)

Spezifikation nach arc42 Quality Model:
„Enable users to perform their tasks safely, effectively and efficiently while enjoying the experience“
(Starke, arc42 Quality Model, 2022)

Der Anspruch an die Gesamt-Lösung und somit auch an die Middleware ist es, dass der Kunde Freude hat das Produkt einzusetzen. Zudem soll es ihm einen Mehrwert gegenüber den jeweiligen proprietären Lösungen bieten. Es soll nicht nur eine Sammlung derselben Funktionalitäten der

bisherigen plattform-spezifischen Lösungen sein, sondern zusätzlich zu diesen noch weitergehende Vereinfachungen, Individualisierungen oder Effizienzsteigerungen bieten.

1.6.1.3. Flexibilität (flexibel)

Spezifikation nach arc42 Quality Model:
„Serve a different or expanded set of requirements, the ease with which the product can be adapted to changes in its requirements, contexts of use, or system environment. Synonyms: modifiable, adjustable, changeable, versatile“
(Starke, arc42 Quality Model, 2022)

Das Gesamtsystem soll unabhängig von den zugrundeliegenden Protokollen, Modulen oder Plattformen gleich funktionieren. Es soll für den Endbenutzer keinen Unterschied machen, ob die Information oder das Signal auf einem offenen standardisierten Protokoll oder auf dem proprietären Produkt eines Herstellers übertragen wird.

1.6.1.4. Bedienbarkeit (operable)

Spezifikation nach arc42 Quality Model:
„Easy to deploy, operate, monitor and control“ (Starke, arc42 Quality Model, 2022)

Die Bedien- oder Abfrage-Inputs der Endbenutzer sollen sich natürlich anfühlen und über die verschiedenen Altersgruppen intuitiv sein. Das Einrichten, Betreiben und Unterhalten soll für den User so einfach wie möglich sein.

Im Rahmen dieser Arbeit beschränkt sich diese Anforderung darauf, dies den User-Interface Entwicklern zu ermöglichen, indem höchstmögliche Flexibilität in Bezug der User-Interface-API abfragen geboten wird. Zudem wird eine erste Demo User-Interface Applikation erstellt und diese von Endbenutzer und von Software-Entwicklern getestet.

2. Randbedingungen

Die Randbedingungen sollen den Herstellern der Smart Home- und IoT-Systeme und den User-Interface Entwicklern eine auf der technischen Ebene basierende Übersicht über die Bedingungen geben, unter welcher der POC erstellt wurde.

2.1. Technische Randbedingungen

In diesem Abschnitt werden die technischen Rahmenbedingungen beschrieben, die für das Projekt relevant sind. Diese Randbedingungen können Einfluss auf die Architekturentscheidungen und die Umsetzung der Software haben. Sie bilden die technischen Leitplanken bei der Entwicklung.

2.1.1. Technische Randbedingungen für Konnektoren

- **Programmiersprachen-Unabhängigkeit**
  Die Konnektoren müssen so entwickelt werden, dass sie unabhängig von der verwendeten Programmiersprache funktionieren. Dies stellt sicher, dass sie in verschiedenen Umgebungen und mit unterschiedlichen Technologien problemlos integriert werden können. Eine plattformunabhängige Gestaltung der Konnektoren gewährleistet maximale Flexibilität und Kompatibilität.

- **Verbindung**
  Die Konnektoren müssen die Kommunikation mit dem Fremdsystem aufrechterhalten, auch wenn die Verbindung zur Middleware kurzzeitig unterbrochen ist. Eine vorübergehende Unterbrechung der Cloud-Verbindung darf keine Funktionsstörungen oder Kommunikationsverluste mit dem Fremdsystem verursachen.

- **Nicht-eingreifende Integration der Konnektoren**
  Die Konnektoren müssen so gestaltet sein, dass sie die Fremdsysteme nicht eingreifend beeinflussen oder verändern. Die Fremdsysteme müssen auch ohne die Konnektoren weiterhin voll funktionsfähig sein. Die Konnektoren dürfen keine dauerhaften Änderungen oder Abhängigkeiten in den Fremdsystemen erzeugen.

2.1.2. Middleware

- **Cloud-Fähigkeit**
  Der Betrieb der Middleware soll in der Cloud, konkret in der Azure-Cloud von Microsoft, möglich sein. Zusätzlich muss die Middleware so gestaltet sein, dass sie auch auf anderen Cloud-Anbietern, wie AWS und Google Cloud, betrieben werden kann. Dies gewährleistet Flexibilität und verhindert eine Abhängigkeit von einem einzelnen Cloud-Dienstleister.

  Skalierbarkeit
  Die Middleware muss in der Lage sein, mit wachsenden Datenmengen und steigender Anzahl an Benutzeranfragen effizient umzugehen. Sie soll horizontal und vertikal skalierbar sein, um sowohl die Leistung als auch die Verfügbarkeit zu gewährleisten.

Latenz
Die Middleware muss eine hohe Leistung bieten, um niedrige Latenzzeiten und schnelle Verarbeitungsgeschwindigkeiten sicherzustellen. Dies bedeutet, dass die Middleware optimierte Algorithmen und Datenstrukturen verwenden sollte, um effiziente Datenverarbeitung und -übertragung zu gewährleisten.

2.1.3. Messaging

- **JMS**
  Die einzusetzende Messaging-Lösung soll den JMS-Standard unterstützen, um eine breite Kompatibilität mit bestehenden und zukünftigen Java-basierten Anwendungen zu gewährleisten. Dies ermöglicht eine einfache Integration und Austauschbarkeit von Komponenten. Der JMS-Standard bietet zudem eine robuste und bewährte Grundlage für zuverlässige Nachrichtenübermittlung.

- **Skalierbarkeit**
  Um bei zunehmenden Benutzern innert wenigen Stunden die gewohnten Reaktionszeiten der Übertragung wieder herstellen zu können, soll die Messaging-Lösung einen starken Fokus auf die schnelle und skalierbare Übertragung legen.

2.2. Organisatorische Randbedingungen

- **Team**
  Das aktive Team dieser Arbeit besteht einzig aus einer Person. Zudem umfasst das verfügbare Zeitbudget dieser Arbeit 360-Stunden. Daher ist es nicht möglich an allen Stellen eine breite Funktionalitäts-Palette zu liefern.

- **Zeitplan**
  Der Start war am 19.02.2024 und die offizielle Abgabe der Arbeit ist am 14.06.2024.

- **Vorgehen**
  Die Entwicklung und Planung geschehen im Scrum-Modell nach dem Vorbild der interaktiven Entwicklung. Bei der Dokumentation wird das branchenbekannte Arc42-Modell als Vorlage zur Unterstützung genommen.

- **Entwicklungswerkzeuge**
  Für die Programmierung des Codes kommt IntelliJ IDEA als Entwicklungsumgebung zum Einsatz. Ausserdem werden die Programme in Java mithilfe der Framework-Bibliothek Spring geschrieben. Die Applikationen werden mit Maven gebaut und dann eigenständig, unter der Hilfestellung der Java-Runtime, lauffähig sein.

2.3. Konventionen

- **Sprache**
  Die Sprache, in der der Code geschrieben wird, ist Englisch. Auch sind die Bezeichnungen auf dem Demo-User-Interface in der englischen Sprache abgefasst. Die Dokumentation wurde bewusst in Deutsch verfasst, da sich die Zielgruppe in der DACH-Region befindet.

- **KNX-Swiss Richtlinien**
  KNX-Projekt-Architektur muss den Richtlinien von KNX-Swiss entsprechen. Diese Richtlinien sind eine Abbildung von Best-Practices innerhalb der DACH-Region.

3. Kontextabgrenzung

Im Gegensatz zum Kapitel 2, indem es vorwiegend um die allgemeinen und Organisatorischen Bedingungen geht, betrifft dieses Kapitel die fachlichen Bedingungen unter welcher der PoC entwickelt wurde.

3.1. Fachlicher Kontext

Um die Herausforderungen umfassend zu verstehen ist es entscheidend, die technischen Unterschiede zu ergründen, die von den Fremdsystem-Herstellern umgesetzt werden. Daher beschäftigt sich der folgende Abschnitt mit den zugrunde liegenden Grundfunktionalitäten der Smart Home- und IoT-Systeme.

3.1.1. Einsatzbereiche der Smart Home und IoT-Komponenten

Heute werden viele klassische, elektromechanische Schalter und Sensoren durch intelligente Smart Home und IoT-Komponenten abgelöst. Diese Ablösung erfolgt in den unterschiedlichsten Gewerken des Neubaus aber auch bei Renovationen bestehender Gebäude, in den unterschiedlichsten funktionale Einsatzbereichen.

Für die Umsetzung dieser Arbeit ist es wichtig, die unterschiedlichen Schalt- (Aktoren) und Sensorkomponenten in ihren Grundfunktionalitäten zu betrachten.

3.1.2. Elektrotechnische Definitionen

Bei den elektrotechnischen Definitionen geht es darum, exemplarisch die Basiskomponenten eines Schalters und eines Tasters zu analysieren. In der Vergangenheit wurden bei Haus- und Industrieinstallationen meist mechanische Schalter oder Taster eingesetzt. In der Analyse der Grundfunktionen geht es darum, ihre expliziten (z.B. das Schalten) und impliziten (z.B. persistente Erhaltung eines Schaltzustandes) Funktionskomponenten herauszuarbeiten. Dabei gilt es zu analysieren, was die Grundfunktion ist, jedoch auch was nicht Bestandteil der Grundfunktion ist.

3.1.2.1. Schalter

Aus elektrotechnischer Sicht schliesst ein Schalter einen elektronischen Strom-Kreislauf. Dabei ist es unerheblich wie lange der Schalter aktiv betätigt wird, der Strom fliesst so lange bis der Schalter wieder physisch umgelegt wird.

Bei einem Schalter gibt es zwei Ausprägungen:

3.1.2.1.1. 1. Hauptausprägung: Wechsel-Schalter

Ein Wechselschalter öffnet beim Schalten den einen Stromkreislauf und schliesst gleichzeitig, bzw. kurz darauf, den anderen Strom-Kreislauf. Bei mechanischen Wechselschaltern ist es nicht möglich, dass beide Ausgänge gleichzeitig geschlossen sind (gegenseitige Verriegelung).

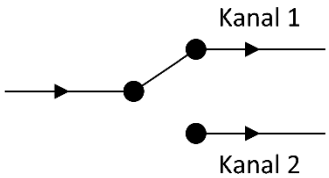


Abbildung 2: Wechsel-Schalter

Grundfunktionen:

- Schalten Kanal 1 (EIN /AUS)
- Schalten Kanal 2 (AUS / EIN)
- gegenseitige Verriegelung
- persistente Erhaltung des Schaltzustandes

Nicht Teil der Grundfunktion bzw. nicht möglich:

- beide Kanäle gleichzeitig EIN oder AUS

Bezug zu den Haushalten: Wechselschalter für Jalousien
Bei einer Jalousie wird derselbe Motor für das Hoch- und Runterfahren der Jalousien in die jeweils andere Richtung laufen gelassen. Dies wird dadurch erreicht, dass der Strom-Kreislauf (z.B. 1-2, Wicklung für Drehrichtung AUF) geöffnet und für die der anderen Richtung (1-3, Wicklung für Drehrichtung AB) geschlossen wird. Gleichzeitig muss sichergestellt sein, dass immer nur ein Stromkreis geschlossen ist.

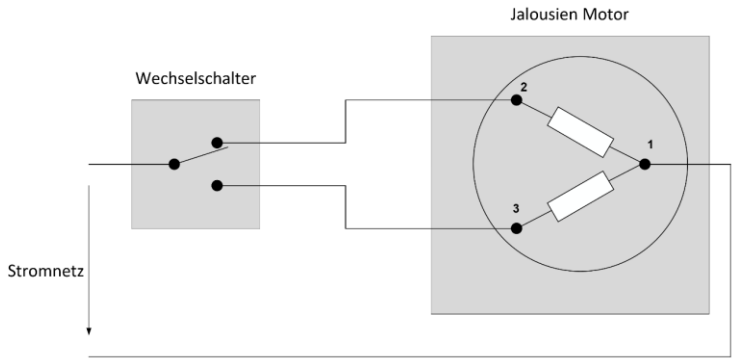


Abbildung 3: Elektrotechnisches Schema eines Jalousie-Motors

3.1.2.1.2. 2. Hauptausprägung: Ein/Aus-Schalter

Beim Ein/Aus-Schalter öffnet oder schliesst sich der Strom-Kreislauf ohne Beeinflussung eines anderen Strom-Kreislaufes. Aus "Steuerungssicht" wandelt ein Schalter einen Input "das eigentliche Schalten" in einen persistenten Schaltzustand um.

Abbildung 4: Ein/Aus Schalter

Grundfunktionen:
- Schalten Kanal 1 (EIN /AUS)
- persistente Erhaltung des Schaltzustandes

Bezug zu den Haushalten
In den früheren Jahren wurden in den Gebäuden jeweils Schalter verbaut, die den elektrotechnischen Schaltern entsprachen. Die Zuleitungen der Verbraucher, wie zum Beispiel von Lampen, wurden von den Schaltern physikalisch geschlossen bzw. unterbrochen.

3.1.2.2. Taster

Ein Taster aus der Sicht der Elektrotechnik bezeichnet eine Vorrichtung, die den Strom-Kreislauf so lange schliesst oder öffnet, wie die aktive Betätigung des Tasters gegeben ist. Auch hier gibt es zwei unterschiedliche Ausprägungen:

3.1.2.2.1. 1. Ausprägung: Schliesser

Beim Schliesser wird der Strom-Kreislauf während der Dauer der physischen Betätigung des Tasters geschlossen. Aus Steuerungssicht generiert ein Taster einen Impuls, wandelt diesen jedoch nicht in einen persistenten Schaltzustand um.

Abbildung 5: Schliesser

Grundfunktionen:
- Schalten Kanal 1 (EIN)
  Eine Aktion (drücken, Input) ergibt immer einen definierten Schaltzustand (AUS)

Nicht möglich:
- persistente Erhaltung des Schaltzustandes

Bezug zu den Haushalten:
In modernen Haushalten, welche ein Gebäudeleitsystem oder funktionsspezifische Steuerungskomponenten enthalten, werden heute meist Taster der Ausprägung Schliesser verbaut. Diese dienen als Inputgeber für die Steuerungssensoren. Das Schalten des Verbrauchers und das Erhalten eines persistenten Schaltzustands (das eigentliche Schalten), wird durch die Steuerung übernommen.

Der Vorteil ist, dass so für einen erzeugten Input (z.B. Drücken eines Tasters) ein oder mehrere, unterschiedliche Schaltvorgänge programmiert werden können.

3.1.2.2.2. 2. Ausprägung: Öffner

Bei einem Öffner unterbricht der Taster während der Betätigungsdauer den Strom-Kreislauf. Hierbei wird eine Feder zusammengedrückt, die ohne aktive Betätigung den Strom-Kreislauf schliesst.

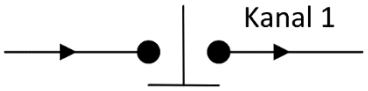


Abbildung 6: Öffner

Grundfunktionen:

- Schalten Kanal1 (AUS)
  Eine Aktion (drücken, Input) ergibt immer einen definierten Schaltzustand (AUS)

Nicht möglich:

- persistente Erhaltung des Schaltzustandes

Bezug zu den Haushalten:
Wie aus dem «Bezug zu den Haushalten» des vorangegangenen Unterabschnittes 3.1.2.2.1 - 1. Ausprägung: Schliesser hervorgeht, kommen Öffner heutzutage nur noch selten zur Anwendung.

3.1.3. Logische Interpretationen

Dieser Unterabschnitt behandelt die logischen Interpretationen der vorhergehenden elektrotechnischen Begriffe und Konzepte.

3.1.3.1. Digitalisierung der Input-Signale

Mit der zunehmenden Vernetzung der Heim-Systeme müssen mechanische Inputs (betätigen eines Schalters) oder elektrische Schaltzustände (z.B. Taster öffnen) digitalisiert werden, um eine Übertrag- und Steuerbarkeit zu erlangen. Dies wird durch die Detektion der Über- und/oder Unterschreitung eines elektrischen Schwellwertes realisiert. Die Interpretation des Input Signals geschieht anschliessend in der digitalisierten Version des Signales. D.h. die logische Schaltfunktion, welche durch ein digitalisiertes Inputsignal ausgelöst wird, (z.B. durch das Drücken eines Tasters), wird durch die Funktionslogik erzeugt.

Digitalisierung des Signales
Bei der Digitalisierung von Inputsignalen/ Schaltzuständen sind zwei Situationen zu berücksichtigen.

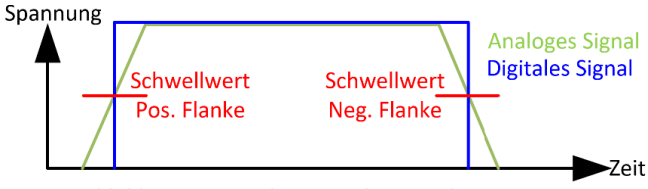


Abbildung 7: Digitalisierung des Signales

1. Überschreitung des elektrischen Schwellwertes => positive Flankentriggerung
2. Unterschreitung des elektrischen Schwellwertes => negative Flankentriggerung

Für diese Arbeit werden die drei folgenden Standard-Funktionslogiken festgelegt:

- UM-Schalten (toggle)
- EIN-Schalten
- AUS-Schalten

3.1.3.1.1. Um-Schalten

Die logische Schaltfunktion «UM-Schalten» (toggle), die ein entsprechendes digitalisiertes Signal oder einen entsprechenden Software-Befehl als «UM-Schalten» interpretiert, versteht darunter das Wechseln / Umschalten des aktuellen, binären Schaltzustands. Dies kann von einem Ein- zu einem Aus-Zustand sowie vom Aus- zu einem Ein-Zustand sein. Wenn also ein logischer Um-Schalter dreimal nacheinander betätigt wird, so sieht die logische Interpretation, beim aktuellen Status «Aus», wie folgt aus: «Ein», «Aus», «Ein».

3.1.3.1.2. Ein/Aus-Schalten

Die logische Schaltfunktion «EIN-Schalten» bzw. «AUS-Schalten», die ein entsprechendes digitalisiertes Signal oder einen entsprechenden Software-Befehl als «EIN-Schalten» bzw. «AUS-Schalten» interpretiert, versteht darunter die Erstellung eines fixen Schaltzustandes (EIN / AUS), unabhängig vom aktuell existierenden Schaltzustand. Dies kann von einem «AUS» zu einem «EIN» Zustand, sowie vom «EIN» zu einem «EIN» Zustand sein.

Wenn also ein logischer EIN-Schalter dreimal nacheinander betätigt wird, so sieht die logische Interpretation, beim aktuellen Status «Aus», wie folgt aus: «EIN», «EIN», «EIN». Daher muss der aktuelle Status nicht gespeichert werden.

3.2. System-Kontext

Der folgende Abschnitt befasst sich mit den von den Fremd-Systemen zur Verfügung gestellten Informationen und Schnittstellen.

3.2.1. Shelly-Kontext

Wenn in den folgenden Unterabschnitten auf alle Shelly Geräte verwiesen wird, so umfasst dies alle Shelly Geräte der zweiten und dritten Generation.

RPC

Shelly bietet ihren Benutzern an, gewisse Prozeduren auf den Geräten auszuführen. Dabei entspricht jede Prozedur einer Methode auf dem entsprechenden Shelly Gerät. Die Übergabe-Parameter und die Methoden-Antwort werden mittels JSON-Objekte übergeben. Das zur Kommunikation verwendete Transportprotokoll ist HTTP. Shelly unterstützt auch die Verwendung von SSL-Zertifikaten. *Diese wurden jedoch im Rahmen dieser Bachelorarbeit nicht berücksichtigt. Weitere Informationen diesbezüglich stehen im ADR 9.1.6 - Keine serverseitige Zertifikatsüberprüfung bei Shelly-Geräten beschrieben.*

WebSockets

Externe Geräte können sich auf den Shelly-Geräten via WebSockets auf Änderungen registrieren. So werden diese externen Geräte über alle Änderungen, welche auf dem jeweiligen Gerät geschehen, mit an das externen Geräte übermittelt. Nachfolgend ist ein Auszug eines Beispiels das veranschaulicht, wie der zeitliche Ablauf und der Inhalt eines Schaltvorganges aussieht, dargestellt.

Sensor-Gerät: Ein

```
{
  "src": "shellyplusi4-d4d4da3ae24c",
  "dst": "Shelly-SmartBridge-
          Connector",
  "method": "NotifyStatus",
  "params": {
    "ts": 1717747427.38,
    "input:3": {
      "id": 3,
      "state": true
    }
  }
}
```

Aktor-Gerät: Ein

```
{
  "src": "shellypro4pm-c8f09e87e46c",
  "dst": "Shelly-SmartBridge-
          Connector",
  "method": "NotifyStatus",
  "params": {
    "ts": 1717747427.49,
    "switch:1": {
      "id": 1,
      "output": true,
      "source": "HTTP_in"
    }
  }
}
```

Sensor-Gerät: Aus

```
{
  "src": "shellyplusi4-d4d4da3ae24c",
  "dst": "Shelly-SmartBridge-
          Connector",
  "method": "NotifyStatus",
  "params": {
    "ts": 1717747427.62,
    "input:3": {
      "id": 3,
      "state": false
    }
  }
}
```

Hervorzuheben ist dabei, dass es sich bei der Übermittlung der Nachricht jeweils nur um die Sicht und den Informationsstand des jeweiligen Geräts handelt. Daher kann das Aktor-gerät nur mitteilen, dass dessen Schalt-Befehl über http übertragen wurde, nicht jedoch woher der Befehl kam. Auch beim Sensor-Gerät ist zu bemerken, dass sich zum Zeitpunkt des Loslassens des Tasters, eine erneute Nachricht versendet wird. Dies basiert darauf, dass die Statusänderungen noch nicht interpretiert werden sondern als Änderung des Status angesehen und versendet werden.

3.2.1.1. Shelly-API

Da der Funktions-Umfang der einzelnen Shelly-Geräte unterschiedlich ist, bietet jede Komponente einen Endpunkt der die Aufgabe hat, alle unterstützten Informationen mitzuteilen. Ausserdem kann auch der aktuelle Status aller Teilsysteme eines Gerätes abgefragt werden.

3.2.1.2. System-API

Um spezifisch und detailliert über den Status eines Geräts informiert zu werden, bietet die Endpunkt-Gruppe System die Möglichkeit, den Status abzufragen sowie die Einstellungen einzusehen und anzupassen.

3.2.1.3. Outbound-Websocket-API

Um in real-time über Statusänderungen auf den Shelly-Geräten informiert zu werden, bieten diese eine Websocket-Verbindung an. Diese bleibt bestehen, bis der Client das Ende der Verbindung verlangt.

3.2.1.4. Cover-API

Da die Steuerung von Jalousien und ähnlichen Geräten desselben Gewerkes spezielle elektrotechnische Schaltungen erfordern, abstrahiert diese Shelly und stellt den API-Anwendern die abstrahierten Funktionalitäten zur Verfügung. Daher bieten sie eigene Endpunkte, um die Jalousien zu steuern, ohne den elektrotechnischen Hintergrund kennen zu müssen.

Weitere Informationen bezüglich des erwähnten elektrotechnischen Hintergrundes ist im Unterabschnitt 3.1.2.1.1 - 1. Hauptausprägung: Wechsel-Schalter beschrieben.

3.2.1.5. Input-API

Ein Input aus Sicht der Shelly-Geräte kann sowohl digital, das heisst via http-Verbindung als auch analog, via Strom-Sensor innerhalb des Geräts, sein. Wenn das Input-Signal von einer anderen Shelly-Komponente kommt, so geschieht dieses auch über das RPC-Protokoll.

Die Unterscheidung, ob es sich um ein analoges oder ein digitales Signal handelt, wird bei der Websocket-Übertragung mittels eigenen JSON-Attributs bewerkstelligt.

3.2.1.6. Switch-API

Ein Schalter im Kontext der Shelly-Komponenten ist eine Ausgangskomponente, welche den Relais-Kontakt schliesst. Hierbei kommen im Wesentlichen zwei Haupt-Prozeduren zum Einsatz.

Prozedur 1: Das Relais wird geschlossen oder geöffnet und
Prozedur 2: Das Relais wird in den umgekehrten Zustand versetzt.

3.2.2. Mapping im Middleware-Kontext

Senden von digitalen Input-Signalen

Eine jener Herausforderungen beim Erstellen des Anti-Corruption-Layers zwischen dem Bounded-Context von Shelly und dem der SmartBridge war die Tatsache, dass Shelly bei den WebSocket-Nachrichten von den Switch-Endpoints einzig mitliefert, ob die Nachricht von einem analogen oder

digitalen Signal getriggert wurde. Es ist auch nicht möglich sich explizit nach dem Absender der http-Nachricht (digitales Signal) zu erkundigen.

Der Absender der Nachricht kann nur ermittelt werden, wenn alle Input-Geräte, welche in der letzten Zeit ein Signal gesendet haben, überprüft werden, ob diese in ihrem Speicher eine Aktion hinterlegt haben, bei welcher sie den entsprechenden Switch-Endpoint aufrufen.

Dies funktioniert jedoch nur, wenn das physische Shelly-Input-Gerät auch beim Shelly-Konnektor registriert ist.

Auto-Discover
Der Shelly-Konnektor verfügt über eine Auto Discovery Funktion, welche alle IP-Adressen innerhalb desselben Netzwerkes parallel zueinander, auf die RPC-Prozedur «GetDeviceInfo» abfragt. Wenn die angefragte IP-Adresse eine Antwort liefert, welche einer Shelly-Nachricht gleichkommt, so wird dies dem Endbenutzer mit IP-Adresse und der Shelly-eindeutigen Gerätebezeichnung aufgelistet.

3.2.3. KNX-Kontext
KNX ist ein weltweit anerkannter Standard für die Gebäudeautomation, der die Steuerung und Vernetzung verschiedener Geräte wie Beleuchtung, Heizung, Lüftung und Sicherheitssysteme ermöglicht. Er basiert auf einem dezentralen Ansatz, bei dem alle angeschlossenen Geräte miteinander kommunizieren können, ohne dass ein zentraler Controller erforderlich ist.

3.2.3.1. KNX-Projekt (ETS-Projekt)
Um bei einem digitalen Zwilling die Geräte auch an ihrem zugehörigen Platz zu finden, kann das gesamte Gebäude abgebildet werden. So können für jedes Projekt folgende Gebäude-Hierarchien geschachtelt erstellt werden:

- Gebäude
- Gebäudeteile
- Etagen
- Räume

3.2.3.2. Geräte
Jeder Hersteller, welcher das KNX-Protokoll unterstützt, muss für alle seine Geräte eine KNX-Produkt-Datei zur Verfügung stellen. Diese kann in die ETS-Software importiert werden und so der digitale Zwilling des korrekten Geräts platziert werden.

Auf diesem Gerät kann dann die Konfiguration vorgenommen werden. Die Parametrisierung, wie zum Beispiel die Betriebsart der Aktoren geschieht über den Reiter Parameter. Basierend auf diesen Parametern können unterschiedliche Kommunikationsobjekte hinterlegt werden.

Parameter
Da viele Hersteller nicht für jeden Anwendungsfall ein eigenes Gerät in ihrem Sortiment haben, produzieren sie Kombi-Geräte, die für den jeweiligen Anwendungsfall konfiguriert werden können.

Kommunikationsobjekte
Die Kommunikationsobjekte sind diejenigen Daten-Elemente, welche über den Bus versendet werden und die entsprechenden Informationen enthalten. Sie sind die physische Repräsentation der logischen Adressen. Sie enthalten einen Datentyp, welcher für die korrekte Interpretation der Codierung benötigt wird. Zudem enthalten sie die Adressen, so dass alle Geräte, welche am Bus angeschlossen sind, entscheiden können, ob sie auf das Kommunikationsobjekt reagieren oder nicht. Dies geschieht sobald in deren Adress-Tabelle die Adresse registriert wurde.

3.2.3.3. KNX-Telegramm

Da KNX auf der Bus-Technologie basiert, enthalten die versendeten Telegramme alle Informationen, welche die Teilnehmer benötigen, die darauf reagieren wollen. So setzt sich ein Datenpaket auf dem KNX-Bus ausfolgenden Elementen zusammen:

Inhalt

-	Steuerfeld	1 Byte	Priorisierung der Nachricht
-	Quelladresse	2 Byte	Physikalische Adresse des Senders
-	Zieladresse	2 Byte + 1-bit	Gruppen-Adresse der Empfänger
-	Routing-Zähler	3-bit	Abnehmender Hop-Counter von 7
-	Länge der Nutzinformation	4-bit	Länge der Nutzinformation in Byte
-	Nutzinformation	1-bit – 14 Byte	Information des Schaltbefehls
-	Sicherungsfeld	1 Byte	Kontrolle der Übertragung

Quelladresse

Die Quelladresse ist die physikalische Adresse des Absenders, welcher die Nachricht auf den Bus geschrieben hat. Dies ist meistens ein Sensor, welcher die interessierten Systemteilnehmer über eine Zustandsänderung informieren möchte. Den Sender der Nachricht interessiert es nicht, wie viele Empfänger die Nachricht lesen.
Weitere Informationen sind im Unterschnitt 3.2.3.3.2 - Physikalische Adressen beschrieben.

Zieladresse

Die Zieladresse ist jeweils eine Gruppenadresse auf die interessierte Systemteilnehmer hören, wenn sie über eine Zustandsänderung informiert werden möchten.
Weitere Informationen sind im Unterschnitt 3.2.3.3.1 - Gruppen Adressen beschrieben.

Nutzinformation

Die Nutzinformationen, welche im KNX-Telegramm verpackt sind wurden standardisiert, um die Funktionalität zwischen verschiedenen Herstellern sicherzustellen.
Weitere Informationen sind im Unterschnitt 3.2.3.3.3 - Datenpunkte beschrieben.

Interpretiert sieht dann ein solches Telegramm wie folgt aus:

#	Time	Service	Flags	Prio	Source Address	Source Name	Destination	Destination Name	Hop Count	Type	DPT	Info
3	07.06.2024 09:03:27.881	from bus		Low	1.1.8		1/0/0		6	GroupValueWrite		$00 | Off

Abbildung 8: Screenshot von KNX-Telegrammen

Das Abbild aus einer Monitoring-Software für KNX-Telegramme zeigt das Senden eines Tasters auf, welcher einen Schalt-Aktor erst ein und danach wieder aus geschalten hat. Anhand dieses Telegrammes wird der Unterschied zur Herangehensweise von Shelly gut sichtbar. Denn die KNX-Geräte interpretieren die Statusänderung, in diesem Falle das Betätigen eines Tasters, bereits indem das Sensor-Gerät eine und nicht zwei Nachrichten sendet.

3.2.3.3.1. Gruppen Adressen

Da KNX auf der Bus-Topologie basiert, werden keine Nachrichten zwischen den Komponenten direkt ausgetauscht, sondern es werden logische Adressen versendet, welche codierte Informationen enthalten. Diese Adressen sind durch einen Schrägstrich in drei Hierarchie-Ebenen unterteilt:

Gruppe	**Beispiel**	**Adresse-Range**	**Gruppen-Repräsentation**	**Beispiel**
Hauptgruppe	**1**/X/X	0-31 (5-bit)	Geschoss / Etage	Erdgeschoss
Mittelgruppe	X/**1**/X	0-7 (3-bit)	Gewerk	Beleuchtung
Untergruppe	X/X/**1**	0-255 (8-bit)	Funktion der Verbraucher	Fenster Schlafzimmer auf / zu

Tabelle 2: Aufbau der KNX-Gruppen Adressen

Die Gesamtlänge einer solchen Adresse beträgt 16-bit. Diese können in bis zu 3 Ebenen aufgeteilt werden. Da sich diese Bachelor-Arbeit an die KNX-Swiss-Richtlinien hält, wird nur die weitverbreitetste 3-Ebenen Struktur unterstützt.

Hauptgruppe
Die erste Adress-Ebene ist die Hauptgruppe. Sie repräsentiert die einzelnen Geschosse des Projektes. Die Gesamtlänge der Hauptgruppe beträgt 5-bit und kann somit Werte von 0 - 31 darstellen, was für die meisten Projekte ausreicht.

Mittelgruppe
In einer zweiten Adress-Ebene werden alle Gewerke repräsentiert. Dies wird mittels 3-bit (8 Gewerke) grossen Mittelgruppe bewerkstelligt. Hierbei ist zu erwähnen, dass nur die Mittelgruppen-Variante A nach KNX-Swiss Richtlinie unterstützt wird.

Untergruppe
In der 8-bit grossen Untergruppe können bis zu 256 Funktionen vergeben werden.

3.2.3.3.2. Physikalische Adressen

Zur Identifizierung der einzelnen Geräte werden den Busteilnehmern eindeutige Adressen zugewiesen. Diese 16-bit Adressen bestehen aus drei Teilbereichen, welche durch einen Punkt voneinander getrennt werden.

Adresse	Beispiel	Adresse-Range	Gruppen-Repräsentation	Beispiel
Bereich	**1.**X.X	0-15 (4-bit)	Gebäude	Haus Nord
Linie	X.**1.**X	0-15 (4-bit)	Stockwerk	Erdgeschoss
Teilnehmer	X.X.**1**	0-255 (8-bit)	System-Teilnehmer	Schaltaktor

Tabelle 3: Aufbau der KNX-Physikalischen Adressen

Auch die physikalischen Adressen, die zur eindeutigen Identifikation der System-Teilnehmer verwendet werden, sind in 3-Ebenen unterteilt.

Bereich
Die ersten 4-bit repräsentieren bis zu 16 Bereiche. In einem Zweckbau können dies die verschiedenen Gebäude oder Gebäudeteile sein, wohingegen es bei einem Einfamilienhaus meist nur ein Gebäude und somit einen Bereich gibt.

Linien
Um die System-Teilnehmer auf den entsprechenden Stockwerken zu finden, werden diese mittels der Linien von 0 - 16 (4-bit) durchnummeriert.

System-Teilnehmer
Pro Linie, was im Gebäude ein Stockwerk entspricht, können bis zu 256 (8-bit) Systemteilnehmer eindeutig adressiert werden.

3.2.3.3.3. Datenpunkte

Die Nutzinformationen werden in den KNX-Projekten als Datenpunkte codiert. Somit ist sichergestellt, dass die Komponenten der verschiedenen Hersteller korrekt miteinander kommunizieren können, ohne sich zu kennen.

3.2.3.3.4. Mapping in Middleware- / SmartBridge-Kontext

Im SmartBridge-Kontext hat jede Nachricht einen Sender und einen physischen Empfänger. Dies ist bei KNX nur teilweise der Fall. Dies da der Sender zwar eine physische Adresse ist, jedoch der

Empfänger nur eine virtuelle Gruppen-Adresse darstellt, auf die alle Systemteilnehmer reagieren, wenn sie diese virtuelle Gruppen-Adresse bei sich im Speicher gespeichert haben.

Ausserdem ist es in KNX nicht möglich die Systemteilnehmer abzufragen, auf welche virtuelle Gruppen-Adressen sie wie reagieren.

Basierend auf den obigen Eigenschaften von KNX muss der Endbenutzer dieses Mapping der virtuellen Gruppen-Adressen, über die im Konnektor zur Verfügung gestellten API, erfassen.

3.2.4. Sonos-Kontext

Sonos ist ein Unternehmen, das kabellose Audiosysteme für Heimnetzwerke entwickelt. Mit ihren Lautsprechern können Nutzer Musik in mehreren Räumen gleichzeitig oder in jedem Raum einzeln abspielen. Die Systeme sind einfach zu bedienen und bieten hochwertigen Klang. Sonos ermöglicht die Integration verschiedener Musikdienste wie Spotify, Apple Music und Amazon Music, so dass Nutzer nahtlos auf die Lieblingsmusik zugreifen können.

3.2.4.1. Sonos-Player

Als Player werden die individuellen Lautsprecher bezeichnet, die im Sonos-Audiosystem verwendet werden. Jeder Player kann eigenständig gesteuert werden, was die Wiedergabe, Lautstärke, Equalizer-Einstellungen und Audioquellen umfassten.

3.2.4.2. Sonos-Gruppe

Das Konzept von Gruppen bei Sonos bezieht sich auf die Möglichkeit, mehrere Sonos-Lautsprecher zu einer Einheit zusammenzufassen, so dass sie synchron dieselbe Musik abspielen. Diese Gruppierungen ermöglichen es Nutzern, eine nahtlose Audioerfahrung in mehreren Räumen gleichzeitig zu schaffen.

3.2.4.3. Sonos Control API

Die Sonos Control API ermöglicht die Steuerung und Integration von Sonos-Lautsprechersystemen in andere Anwendungen. Mit dieser API können Funktionen wie Wiedergabesteuerung, Lautstärkeregelung und Gruppenkonfiguration von Sonos-Lautsprechern ferngesteuert werden. Sie bietet spezifische Endpunkte für die Verwaltung von Lautsprechergruppen, was die Synchronisierung der Wiedergabe über mehrere Räume hinweg erleichtert.

Player

Die zuvor erläuterten Speaker können über die API nur in Bezug auf die Laustärke individuell angesteuert werden. Die Operationen, welche im Zusammenhang mit dem Abspielen von Musik stehen, sind nur auf Ebene der Gruppe auszuführen. Basierend auf dieser Eigenschaft wird auch bei einem einzelnen Sonos-Gerät eine Gruppe erstellt.

Groups

Auf jeder Gruppe können einheitlich sowohl die Lautstärke aller enthaltenen Geräte als auch die musik-basierten Operationen gesteuert werden.

3.3. Ubiquitous Language

Das Ziel einer Ubiquitous Language (übersetzt allgegenwärtige Sprache) ist es, ein einheitliches Verständnis zu schaffen und dieses zu dokumentieren, damit alle beteiligten Personen dasselbe unter einem Objekt oder einer Eigenschaft verstehen.

3.3.1. Ubiquitous Language des Middleware-Kontexts

Innerhalb der Middleware hat jedes Moduls seine eigene Ubiquitous Language. Diese stammt bei allen Modulen innerhalb der Middleware von derselben Basis ab. Dies ermöglicht es Objekte in den Modulen individuell persistieren zu können und beim Antworten auf Modul-API Anfragen auf dedizierte DTOs verzichten zu können.

3.3.1.1. Objekt-Identifikator

Um alle Objekte zwischen den verschiedenen Repräsentationen und Instanzen hinaus identifizieren zu können besitzen sie immer einen Objekt-Identifikator. Hierbei handelt es sich um einen Universally Unique Identifier (UUID).

3.3.1.2. User

Ein User ist ein Benutzer, der über ein User-Interface auf die Middleware zugreift.

3.3.1.3. Gerät

Ein Gerät ist ein Objekt, dass eine Aufgabe hat. Das Verständnis eines Gerätes in diesem Kontext ist nicht von der physikalischen Bauweise abhängig. So kann aus Sicht der Middleware-Module eine Baugruppe eines 4-fach Schaltaktors, welche ein physikalisches Gerät mit vier schaltbaren Ausgängen darstellt, als vier unabhängige Geräte angeschaut werden. Dies soll dem Endbenutzer insofern helfen, dass er sich nicht mit der effektiven Verkabelung des Gebäudes beschäftigen muss, um die Vorgänge zu verstehen.

3.3.1.4. Sensor

Da die in der Einleitung dieses Dokumentes beschriebene Verständnis eines Sensors, welche sich das Registrieren einer Änderung des Zustandes und das Mitteilen dieser begnügt, wurde eine weitere Verfeinerung in dieser Ubiquitous Language vorgenommen. Entscheidend für die Unterscheidung ist, wie der gemessene Zustand interpretiert wird. Wenn es sich dabei um einen physikalischen Wert wie eine Temperatur oder die Luftfeuchtigkeit handelt, so wird das Gerät, das diesen Wert an die anderen Geräte übermittelt, als Sensor identifiziert. Handelt es sich jedoch beim interpretierten Wert um einen Schalt-Befehl wie EIN-, AUS- oder UM-Schalten, so wird das Gerät als Befehlsgeber identifiziert. Weitere Informationen zu den Befehlsgebern sind im Unterabschnitt 3.3.1.5 - Befehlsgeber ausgeführt.

Im Rahmen dieser Arbeit wurde die Umsetzung der Sensoren auf die konzeptionellen, theoretischen Arbeiten beschränkt.

Temperatur-Sensoren

Jeder Temperatur-Sensor erfasst die am Standort des Geräts herrschende Temperatur. Diese verfügt jeweils über folgende Informationen:

Information:	***Beispielwerte:***
Messwert	28.5
Masseinheit	Grad Celsius (°C)
Zeitpunkt der Messung	2021-06-14T10:00:00Z

Luftfeuchtigkeits-Sensor

Jeder Luftfeuchtigkeit-Sensor erfasst den am Standort des Geräts herrschenden Prozentsatz des Wasserdampfs in der Luft im Verhältnis zur maximalen Menge, die die Luft bei gleicher Temperatur enthalten kann. Diese Informationen umfassen jeweils die aktuelle relative Luftfeuchtigkeit, die Temperatur am Standort des Sensors sowie den Zeitpunkt der Messung. Jeder Luftfeuchtigkeits-Sensor verfügt über folgende Informationen:

Information:	***Beispielwerte:***
Messwert	81.5
Masseinheit	relative Luftfeuchtigkeit (%)
Zeitpunkt der Messung	2021-06-14T12:00:00Z

3.3.1.5. Befehlsgeber

Der Befehlsgeber versendet, im Gegensatz zu den Sensoren, nicht einen physikalischen Messwert, sondern einen Befehl. Wichtig diesbezüglich ist darauf hinzuweisen, dass der Fokus beim interpretieren Wert liegt. So ist auch ein Gerät, das die Information, dass der Endbenutzer ein Befehl gegeben hat durch eine Spannungs-Messung und dem damit verbundenen Überschreiten des Schwellwertes einer steigenden oder fallenden Flanke, analog aus dem Unterabschnitt 3.1.3.1 - Digitalisierung der Input-Signale gewonnen hat, ein Befehlsgeber.

Befehlsgeber-Ausprägungen:

- Um-Schalten
- Ein / Aus-Schalten
- Start / Pause / Stopp
- Lauter / leiser / lautlos

3.3.1.5.1. Um-Schalten

Die Befehlsgeber der Ausprägung Um-Schalten entsprechen dem logischen Um-Schalter.

3.3.1.5.2. Ein / Aus-Schalten

Die Befehlsgeber der Ausprägung Ein/Aus-Schalten entsprechen dem logischen EIN/AUS-Schalter.

3.3.1.5.3. Start / Pause / Stopp

Um bei einem 3.3.1.6.4 - Multimedia-Aktor die Medien auch laufen zu lassen oder zu unterbrechen sind die Befehle Start, Stopp und Pause von entscheidender Bedeutung.

3.3.1.5.4. Lauter / leiser / lautlos

Neben dem Abspielen der Medien ist auch dessen Regulierung der Lautsprecher von Interesse. Daher wurden die Befehle dieses Befehlssatzes eingeführt.

3.3.1.6. Aktor

Im Gegensatz zu den Sensoren und den Befehlsgebern führen die Aktoren eine Aktion aus. Jalousie-Aktoren können nicht nur Jalousien steuern, sondern unter anderem auch Rollläden, Markisen, Dachfenster, Lichtkuppeln, Projektor-Leinwände oder Sonnenstoren.

Aktor-Ausprägungen:

- Schalt-Aktor *schaltet beispielweise eine Lampe*
- Jalousie-Aktor *lässt beispielweise eine Jalousie hoch oder runterfahren*
- Dimm-Aktor *lässt eine Lampe heller oder dunkler leuchten*
- Multimedia-Aktor *schaltet zum Beispiel eine Musik-Box ein oder aus*

3.3.1.6.1. Schalt-Aktor

Ein Schalt-Aktor kann sowohl Lampen als auch Steckdosen schalten. Zudem umfasst ein Schalt-Aktor folgende Funktionalitäten: Ein- / Aus- / (Um)-Schalten

3.3.1.6.2. Jalousie-Aktor

Obwohl Jalousie-Aktoren aus elektrotechnischer Sicht zwei Schaltaktor-Kanäle sind, die denselben Motor in unterschiedliche Richtungen drehen lassen, unterstützt die Middleware nur explizit für Jalousien konzipierte Jalousie-Aktoren. Diese verfügen über folgende Funktionalitäten: Auf / Ab / Stopp. So wird auch sichergestellt, dass die beiden, aus elektrotechnischer Sicht gesehenen Schaltaktor-Kanäle nicht gleichzeitig aktiv sind (gegenseitige Verriegelung).

3.3.1.6.3. Dimm-Aktor

Dimm-Aktoren steuern die Intensität des Lichts einer Lampe. Dies umfasst folgende Funktionalitäten: Heller / dunkler
Die Implementation vom Dimm-Aktoren ist im Rahmen dieser Arbeit nicht vorgesehen.

3.3.1.6.4. Multimedia-Aktor

Bei Multimedia-Aktoren handelt es sich um Geräte, welche interaktive Medien wie Musik, Bild oder ein Video abspielen können. Diese Geräte implementieren folgenden Funktionsumfang: Start / Stopp / Pause / (Lauter / leiser / lautlos)

3.3.2. Ubiquitous Language der Middleware-Module

Basierend auf den vorangegangenen und in der gesamten Middleware gültigen Definitionen werden diese in den folgenden Unterabschnitten auf die einzelnen Module ausspezifiziert.

3.3.2.1. Ubiquitous Language des Moduls - User-Management

Beim User-Management Modul handelt es sich um das Modul, welches alle relevanten Nutzer-Informationen, inklusive der Einstellungen welche ein Nutzer vornehmen kann, verwaltet.

Gerät im Kontext des User-Managements
Jedes Gerät hat einen Identifikator, einen Namen, eine Beschreibung und Informationen darüber, ob es ein Sensor, ein Sichtschutz oder ein Musikgerät ist. Ausserdem hat jedes Gerät eine Referenz auf den Konnektor, dem es zugeordnet ist. Damit man auch in einigen Jahren noch weiss, wann das Gerät in Betrieb genommen wurde, wird dieses Datum ebenfalls gespeichert.

Konnektor des User-Managements
Jeder Konnektor hat einen Identifikator, eine Namensbezeichnung, eine Beschreibung, einen Typ und einen Besitzer. Ausserdem wird auch das Datum der Inbetriebnahme gespeichert. Der Konnektor dient als Verbindung zwischen dem Benutzer und seinen Geräten.

User im Kontext des User-Managements
Identifiziert wird jeder User über einen eindeutigen Identifikator. Darüber hinaus verfügt er über ein Geburtsdatum, eine E-Mail-Adresse sowie über einen Vor- und Nachnamen. Sein Passwort wird verschlüsselt in der Datenbank gespeichert. Zudem werden auch die Sprache und das Geschlecht abgespeichert.

3.3.2.2. Ubiquitous Language des Moduls - Record-Keeper

Der Fokus des Record-Keeper-Modules liegt auf dem Persistieren der Zustandsänderungen. Daher sind die allgemeinen Informationen auf ein Minimum reduziert, abgestimmt auf den aktuellen Funktionsumfang.

Gerät im Kontext des Record-Keepers
Das Konzept des Gerätes existiert in dem Sinne innerhalb dieses Modules nicht. Einzig der Objekt-Identifikator wird auf dem Konnektor geführt.

Konnektor im Kontext des Record-Keepers
Wie auch die Geräte enthalten die Konnektoren nur die minimal relevanten Informationen, um die Geräte und deren Besitzer verlinken zu können. Dies umfasst neben dem Objekt-Identifikator und dem User-Identifikator noch eine Liste mit den zum Konnektor assoziierten Geräte-Identifikatoren.

User im Kontext des Record-Keepers
Nebst einem Objekt-Identifikator verfügt ein User-Objekt im Rekord-Keeper über eine E-Mail-Adresse und eine Liste an Konnektoren.

Persistence-Trigger Kontext des Record-Keepers
Dem Endbenutzer werden drei verschiedene Granularitätsstufen geboten, aufgrund von denen ein Record persistiert wird. So kann zwischen der Option, dass alle Nachrichten eines Geräts, eines Konnektors (inkl. aller seiner assoziierten Geräte) oder der Option, dass alle Nachrichten des Users (inkl. aller seiner assoziierten Konnektoren und dessen Geräte) persistiert werden, gewählt werden.

Daher verfügt ein Persistence-Trigger in diesem Kontext sowohl über einen Triggertyp, einen Objekt-Identifikator und über eine Trigger-Source, welche den Objekt-Identifikator des entsprechenden Triggertyps enthält.

3.3.2.3. Ubiquitous Language des Moduls - Command-Suite

Innerhalb des Command -Suite Modules sind nur diejenigen Informationen von Relevanz, welche unmittelbar zur Zustandsänderung eines Gerätes benötigt werden.

Gerät im Kontext der Command-Suite
Um auf einem User-Interface die entsprechenden gerätetypspezifischen Funktionalitäten bereitstellen zu können, wird neben dem Namen und dem Objekt-Identifikator auch der Gerätetyp benötigt. Zudem muss das Objekt über einen Besitzer verfügen, um jedem Benutzer nur die eigenen Geräte anzeigen zu können.

Konnektor im Kontext der Command-Suite
Die Command-Suite benötigt vom Konnektor nur deren Objekt-Identifikator zur korrekten Adressierung. Da keine weiteren Informationen des Konnektors benötigt werden ist der Identifikator auf dem Gerät direkt gespeichert.

User im Kontext der Command-Suite
Um den Endbenutzer korrekt ansprechen zu können genügt es neben dem Objekt-Identifikator den Namen und Vornamen des Benutzers zu persistieren. Zudem werden alle Geräte eines Endbenutzers auf dessen Objekt referenziert, um bei einer Abfrage seine assoziierten Geräte schneller zurückliefern zu können.

3.3.2.4. Ubiquitous Language des Moduls - Policy-Hub

Der Policy-Hub stellt eine zusätzliche Automatisierungs-Möglichkeit dar. Dies aus dem Grund, dass die Lebensdauer von heutigen Elektronik-Komponenten im Hausbau von einer durchschnittlichen Lebensdauer von 30 Jahren profitieren und daher noch nicht alle bisher verbauten Elektronik-Komponenten über Automatisierungs-Möglichkeiten verfügen.

Gerät im Kontext des Policy Hubs
Die einzigen gerätespezifischen Informationen, welche für die Automatisierung relevant sind, ist nebst dem Objekt-Identifikator, der Objekt-Identifikator des Konnektors hinter welchem sich, das Gerät befindet.

Konnektor im Kontext des Policy Hubs
Um sowohl Zeit als auch Sonnenstands-basierte Automatisierungen vornehmen zu können sind Zeitzonen und Standort-Informationen von Bedeutung. Da sich die Geräte, die am selben Konnektor angeschlossen sind, ungefähr am selben Standort befinden, werden diese Informationen auf dem Konnektor gespeichert.

User im Kontext des Policy Hubs
Um den User als Reaktion einer Policy zu erreichen, wird neben dem User-Identifikator, sowie den Konnektoren in der Regel auch die E-Mail-Adresse gespeichert.

Regeln im Kontext des Policy Hubs
Die Regeln selbst bestehen aus einem Regel-Identifikator, einer Regel-Bezeichnung, dem Verweis auf den Benutzer dem sie zuzuordnen sind und je einem Verweis auf den Auslöser sowie einer Reaktion.

Regel-Reaktion im Kontext des Policy Hubs
Aktuell sind alle Regel-Reaktionen geräte-basiert. Das heisst, dass es sich bei er Reaktion immer um eine Operation handeln muss, die ein Aktor-Gerät betreffen muss. Das bedeutet, dass die Reaktion immer eine Aktion sein muss, die ein Aktor-Gerät betrifft. Daher wird nebst der Anweisung, was passieren soll, auch das Zielgerät gespeichert. Zudem verfügt eine Regel-Reaktion über einen Objekt-Identifikator und einen Verweis auf die Regel, derer er zugeordnet wird.

Regel-Auslöser im Kontext des Policy Hubs
Auch der Regel-Auslöser basiert auf dem Objekt-Identifikator und einer Referenz auf die Regel selbst. Um sowohl zeit- und datum-basierte als auch sensor-basierte Regel-Auslöser umsetzen zu können, wird nebst einem Auslöser-Typ und dem Auslöser-Wert auch die Information, auf welcher der Auslöser basieren soll, mitgeliefert. Dies soll anhand des folgenden Beispiels veranschaulicht werden**:**

- Auslöse-Typ: «Nachricht vor spezifischer Uhrzeit»
- Auslöse-Quelle: «zeit-basiert»
- Auslöse-Wert: «08:00 Uhr»

3.3.2.5. Ubiquitous Language des Moduls - Monitoring-Suite

Beim Modul Monitoring-Suite werden nicht alle Daten, welche dem Endbenutzer angezeigt werden können, in eine Datenbank persistiert. Dies aus dem Hintergrund, dass es sich bei der Monitoring-Suite um ein Modul handelt, das die letzten Nachrichten, welche auf einem Fremd-System versendet wurden, zwischenspeichert und anzeigen kann. Dies betrifft jedoch nur Geräte, User und Konnektoren, bei welchen dies zuvor explizit hinterlegt worden ist.

Gerät im Kontext des Policy Hubs
Zusätzlich zu den Informationen, ob es sich beim Gerät um einen Sensor handelt und die Geräte-bezeichnung ist die Referenz auf den Konnektor sowie der Geräte-Identifikator von Interesse und werden daher persistiert.

Konnektor im Kontext des Policy Hubs
Einzig der Identifikator des Konnektors und ein Pointer auf den User, dem der Konnektor zuzuordnen ist, sind in diesem Kontext von Interesse.

User im Kontext des Policy Hubs
Der User besteht in diesem Kontext einzig aus seinem Identifikator.

Record im Kontext des Policy Hubs
Bei den Records handelt es sich um eine Übersicht über die wichtigsten Informationen. Dies umfasst neben einem Identifikator und dem Ursprungs-Objekt auch die Zeitpunkte, an denen die Nachricht versendet wurde sowie wann sie auf dem Modul eingetroffen ist.

Hierbei ist jedoch zu beachten, dass diese Informationen nicht persistiert werden sondern nur bis zum nächsten überschreiben einer neueren Nachricht verfügbar sind.

4. Lösungsstrategie

Das Kapitel Lösungsstrategie dokumentiert die Konzepte sowie die Architektur der Arbeit. Es handelt sich somit um ein konzeptionelles Kapitel. Die konkrete Umsetzung der in diesem Kapitel erläuterten Überlegungen sind im darauffolgenden Kapitel 5 - Building Block View dokumentiert.

Daher richtet sich dieses Kapitel an die Hersteller von IoT- und Smart Home-Geräten und an Frontend- (Software)Entwickler.

4.1. Bounded-Context

Ein Bounded-Context grenzt einen Bereich ab, indem eine Ansammlung von Begriffen eine bestimmte Bedeutung besitzen.

Die folgende Grafik gibt einen Überblick über die einzelnen Bounded-Context. So verfügt jedes Fremd-System jeweils über einen eigenen Bounded-Context (querschraffiert dargestellt). Dieser wird von den system-spezifischen Modulen innerhalb der Konnektoren übernommen. Erst beim Versenden der Nachricht an die Middleware wird der Kontext ein erstes Mal gewechselt. Ein weiterer Kontext-Wechsel geschieht, sobald die Daten in die Funktions-Module der Middleware übergehen werden. Diese stellen ihren Kontext (gepunktet dargestellt) nach aussen zur Verfügung.

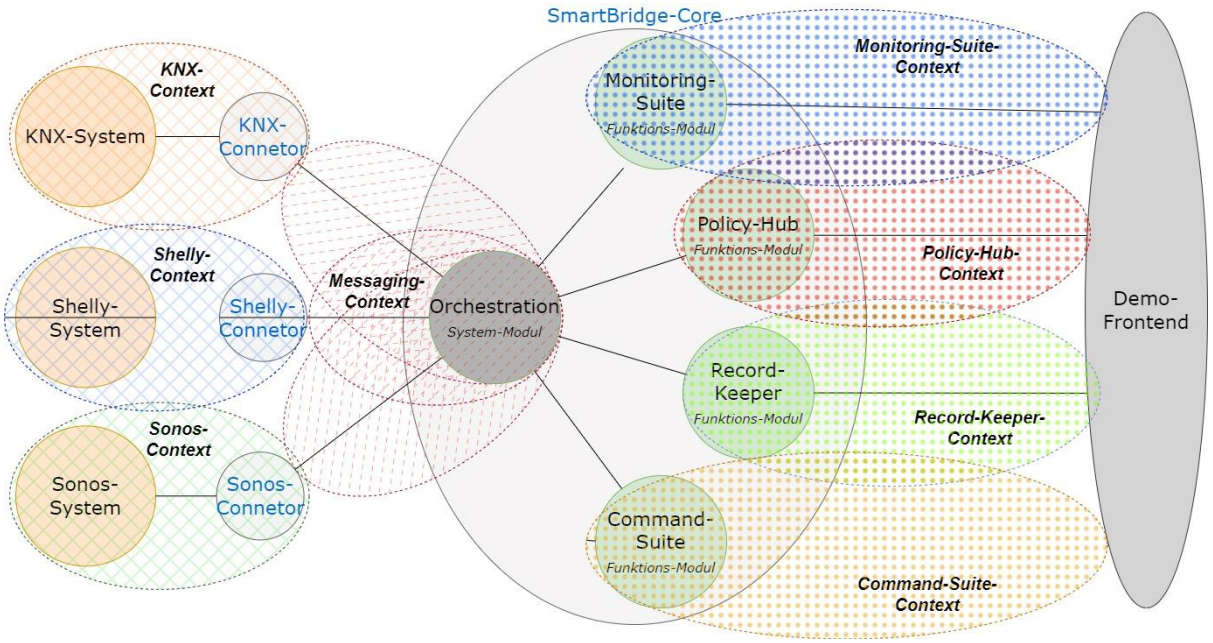


Abbildung 9: Bounded-Context Übersicht

Während die vorangegangene Grafik den Fokus daraufgelegt hat die Kontexte hervorzuheben, liegt der Fokus der folgenden Grafik (Abbildung 10: Bounded-Contexte bezüglich der Abhängigkeiten) mehr auf den Abhängigkeiten zwischen den Modulen.

Hierbei ist zu beachten, dass eine Open-Host-Service Abhängigkeit nicht immer eine API, sondern auch das veröffentlichende Ende eines Messaging-Kanales, welcher auf dem Publish-Subscribe-Pattern (Hohpe & Woolf , Publish-Subscribe Channel, 2003) basiert, sein kann.

In den darauffolgenden Unterabschnitten wird jeweils auf die einzelnen Abhängigkeiten vertieft eingegangen.

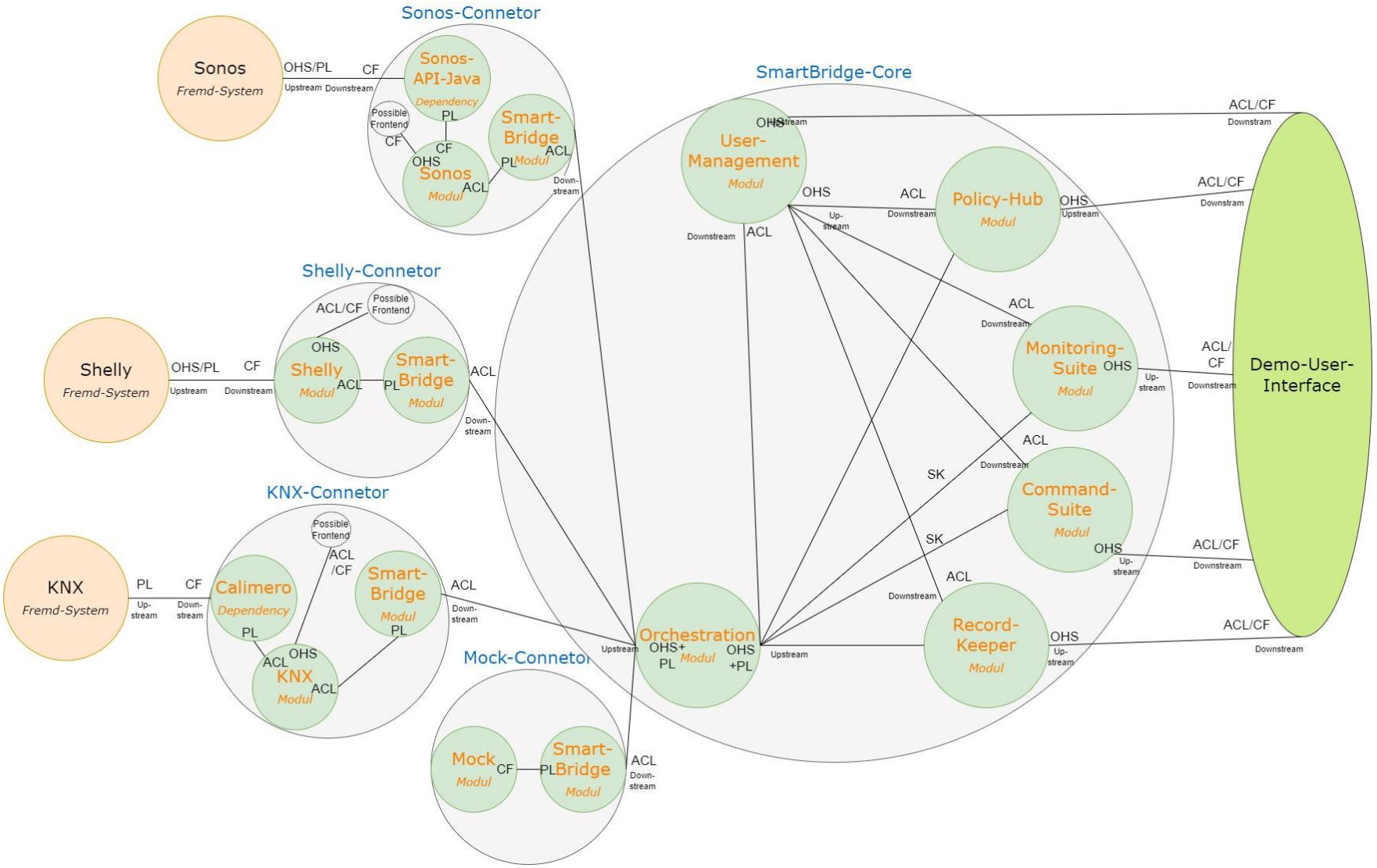


Abbildung 10: Bounded-Contexte bezüglich der Abhängigkeiten

4.1.1. Fremd-Systeme – Konnektoren

Der folgende Unterabschnitt behandelt die Abhängigkeiten aller Fremd-Systeme und deren Konnektoren.

4.1.1.1. Sonos-System – Sonos-Modul

Sonos stellt, anders als andere Fremd-Systeme, die API im Internet zur Verfügung. Das bedeutet, die einzelnen Sonos-Geräte können über die Netzwerk-Grenzen hinaus angesteuert werden. Somit müsste der Sonos-Konnektor nicht zwingend innerhalb desselben Netzwerkes wie die anzusteuernden Sonos-Komponenten sein. Dennoch wurde aktiv entschieden, den Konnektor innerhalb des Netzwerks zu belassen. Eine andere Möglichkeit wäre es in Zukunft auch Cloud-Konnektoren zur Verfügung zu stellen. Diese könnte man als Service anbieten. Sie würden als eigenständige Instanz in der Cloud betrieben werden.

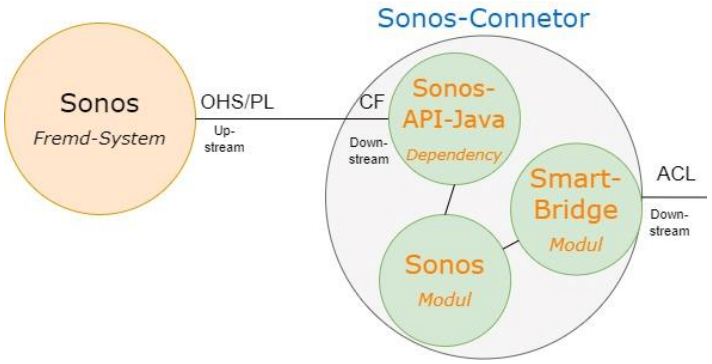


Abbildung 11: Sonos-Konnektor Aussenansicht

Trotz dessen, dass im Umfang dieser Arbeit keine Daten vom Sonos-System in den Bounded-Context von SmartBridge fliessen, musste ein Mapping der Repräsentationen vorgenommen werden. Dies wird für das Ansteuern der Sonos-Komponenten benötigt.

Weitere Informationen sind dem ADR 9.1.9 - Sonos-Konnektor innerhalb desselben Netzwerkes wie die Sonos-Speaker zu entnehmen.

4.1.1.2. Shelly-System – Shelly-Modul

Shelly bietet eine Vielzahl von RESTful-http Web Endpoints auf den Geräten selbst an. Daher unterwirft man sich als Konsument dieser Endpunkte der Public-Language von Shelly.

Aufgrund der Shelly-Kontext-spezifischen Public-Language wurde entschieden beim Austritt der Daten, aus Sicht des Datenflusses in Richtung der Middleware nicht nur die Struktur aufzubrechen, sondern auch mit weiteren Informationen anzureichern und auf die eigene Ubiquitous Language anzupassen.

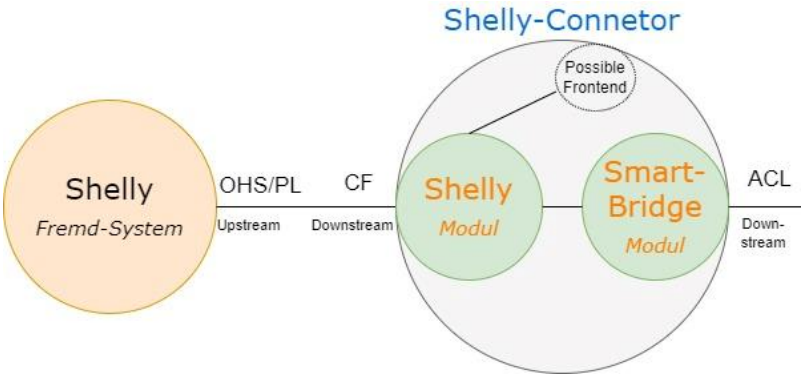


Abbildung 12: Shelly-Konnektor Aussenansicht

4.1.1.3. KNX-System – Calimero

Die Technische Universität Wien hat eine Java-basierte Bibliothek zur Kommunikation mit KNX-Komponenten erstellt. Diese wurde unter der GNU General Public License veröffentlicht.

Basierend auf der Aufgabenstellung, welche nicht die Kommunikation mit den Fremdsystemen selbst sondern die Middleware in den Fokus rückt, wird für die Kommunikation die Java-Library Calimero eingesetzt. Calimero, nimmt jedoch kein Mapping der Daten auf eine eigene Repräsentation vor, sondern stellt sie in derselben Repräsentation wie auf dem KNX-System zur Verfügung. Somit erfüllt die Verbindung zwischen Calimero und dem KNX-System die Anforderungen an einen «Konformisten». *Weitere Informationen bezüglich dieser Entscheidung sind dem ADR 9.1.10 - Calimero als KNX-Kommunikations-Bibliothek zu entnehmen.*

Das Mapping vom KNX-Kontext in den Kontext der Middleware wird im KNX-Modul des KNX-Konnektors vorgenommen.

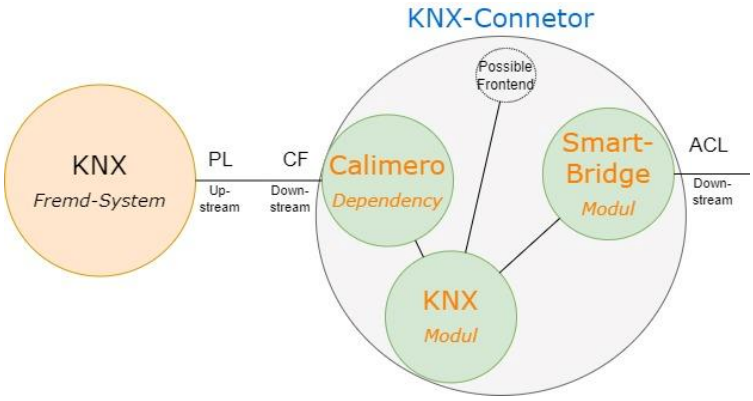


Abbildung 13: KNX-Konnektor Aussenansicht

4.1.2. Konnektoren

Nachdem im obigen Unterabschnitt die Abhängigkeiten der Fremd-System und deren Konnektoren behandelt wurde, werden in diesem Unterschnitt die Abhängigkeiten innerhalb der Konnektoren thematisiert.

4.1.2.1. Kommunikation innerhalb der Konnektoren

Generell wird innerhalb der Konnektoren mittels «Events» kommuniziert. Dies hat den Vorteil, dass die einzelnen Module sich nicht kennen müssen und weitere Module hinzugefügt oder entfernt werden können, ohne auf die bestehenden Module einen Einfluss zu haben.

Die Events werden von dem in Spring-Modulith enthaltenen Application-Event-Publisher veröffentlicht.

4.1.2.2. Sonos-Modul – SmartBridge-Modul

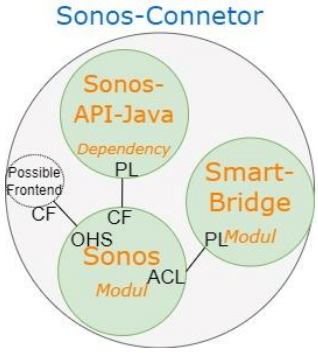


Abbildung 14: Sonos-Konnektor Innenansicht

Wenn eine Message von der Middleware an den Sonos-Konnektor gesendet wird, so trifft sie beim SmartBridge-Modul ein. Hier wird die Nachricht, die JSON-Nachricht de-serialisiert. Anschliessend wird sie vom Application-Event-Publisher innerhalb des Spring-Modulithen (Sonos-Konnektor) veröffentlicht.

Trotz dessen, dass bei einem Status-Update auf dem Sonos-System die Daten nicht via Direkt-Verbindung zwischen Sonos-System und Sonos-Modul in den Konnektor eintreten, sondern via Callback-Modul, wird das Mapping der Repräsentationen auf dem Sonos-Modul vorgenommen.

4.1.2.3. Shelly-Modul – Smart-Bridge-Modul

Sowohl die Anweisungen an die Shelly-Komponenten als auch dessen Status-Updates werden vom Shelly-Modul entsprechend angepasst. Basierend auf der Anforderung, dass Modul SmartBridge konnektor-unabhängig zu entwickeln, wurde seitens Shelly-Modul ein Anti-Corruption-Layer eingebaut. Zudem verfügt auch das Shelly-Modul über die Möglichkeit API-Calls, im boundet-Context des Shelly-Systems, zu senden und zu empfangen.

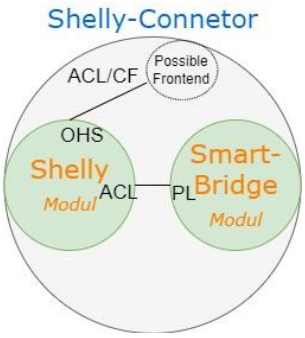


Abbildung 15: Shelly-Konnektor Innenansicht

4.1.2.4. Calimero – KNX-Modul

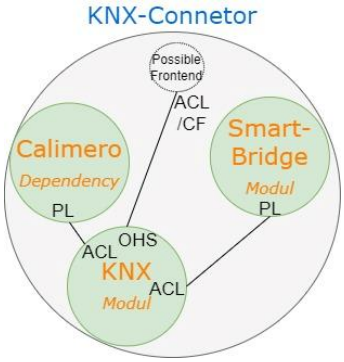


Abbildung 16: KNX-Konnektor Innenansicht

Über Status-Updates innerhalb des KNX-Systems wird die Middleware durch das Interface «ProcessListener», welches das Observer-Pattern implementiert, informiert (Gamma, Helm, Johnson, & Vlissides, 1997). Dies ermöglicht es das Mapping der Repräsentationen innerhalb des KNX-Moduls vorzunehmen.

4.1.2.5. KNX-Modul – Smart-Bridge-Modul

Die Kommunikation zwischen dem KNX-Modul und dem Modul Smart-Bridge erfolgt, analog zum Shelly-Konnektor, auch mit Hilfe des Application-Event-Publishers.

4.1.2.6. Mock-Modul – Smart-Bridge-Modul

Da der Mock-Konnektor die Aufgabe hat ein Fremd-System mittels Konsolen-Applikation zu simulieren, ohne ein physisches Fremd-System betreiben zu müssen, endet der Datenfluss beim Mock-Modul.

Dennoch funktioniert die Kommunikation zwischen dem Mock-Modul und dem Smart-Bridge-Modul analog zu den anderen Konnektoren mittels eventbasierter Kommunikation.

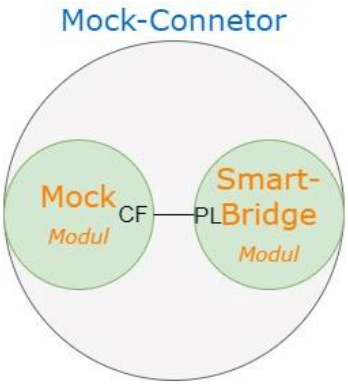


Abbildung 17: Mock-Konnektor Innenansicht

4.1.3. Konnektoren - SmartBridge-Core

Im Folgenden Abschnitt werden die Abhängigkeiten der Konnektoren zur Middleware beleuchtet. Aufgrund der Tatsache, dass das Modul SmartBridge auf den Konnektoren jeweils identisch ist, wird hier ein Konnektor mit dessen Abhängigkeit zur Middleware, exemplarisch für alle aufgeführt.

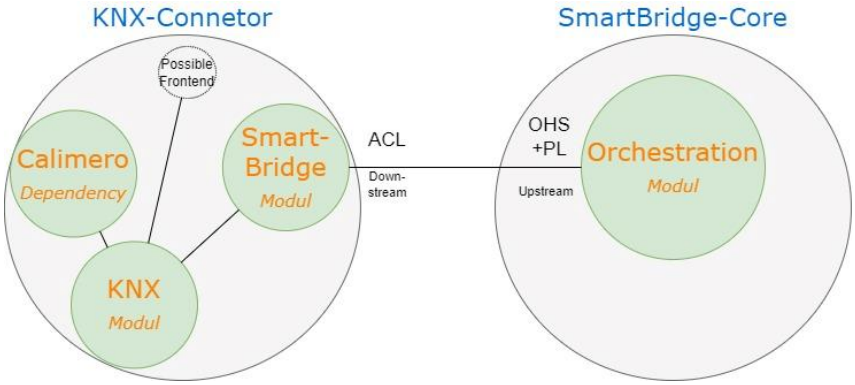


Abbildung 18: Abhängigkeit zwischen Konnektor und Middleware

4.1.3.1. SmartBridge-Modul – Orchestration-Modul

Da der Proof-of-Concept vorsieht, dass die Hersteller der Fremd-Systeme die Konnektoren zur Middleware selbst schreiben, wurde die Kommunikation innerhalb des Konnektors in ein eigenes Modul verschoben. Dieses Modul kann ohne grosse Anpassungen veröffentlicht und anschliessend vom Hersteller der Fremd-Systeme verwendet werden.

Die genaue Definition der Austausch-Nachricht ist im Unterabschnitt 4.2.4 - Messaging Message definiert. Die Hoheit der Definition liegt beim Orchestrations-Modul innerhalb der Middleware. Daher ergibt sich eine OHS – CF Abhängigkeit zwischen den Modulen Orchestration und SmartBridge.

4.1.4. SmartBridge-Core

Jedes der Funktions-Module und das Modul User-Management haben jeweils eigene API-Endpunkte mit eigenen Repräsentationen der Daten. Aus Sicht des API-Benutzers ist dies nicht weiter störend, da die Objekt-Identifikatoren dieselben sind und die Repräsentationen so den Charakter von DTO's haben.

4.1.4.1. Abhängigkeiten zwischen der Middleware und externen Modulen

Die folgenden Unterabschnitte befassen sich mit den Abhängigkeiten, die aus Sicht der Middleware nach aussen gerichtet sind. Hierbei gibt folgende Grafik einen guten Überblick welche Abhängigkeiten unter den Modulen und Instanzen bestehen.

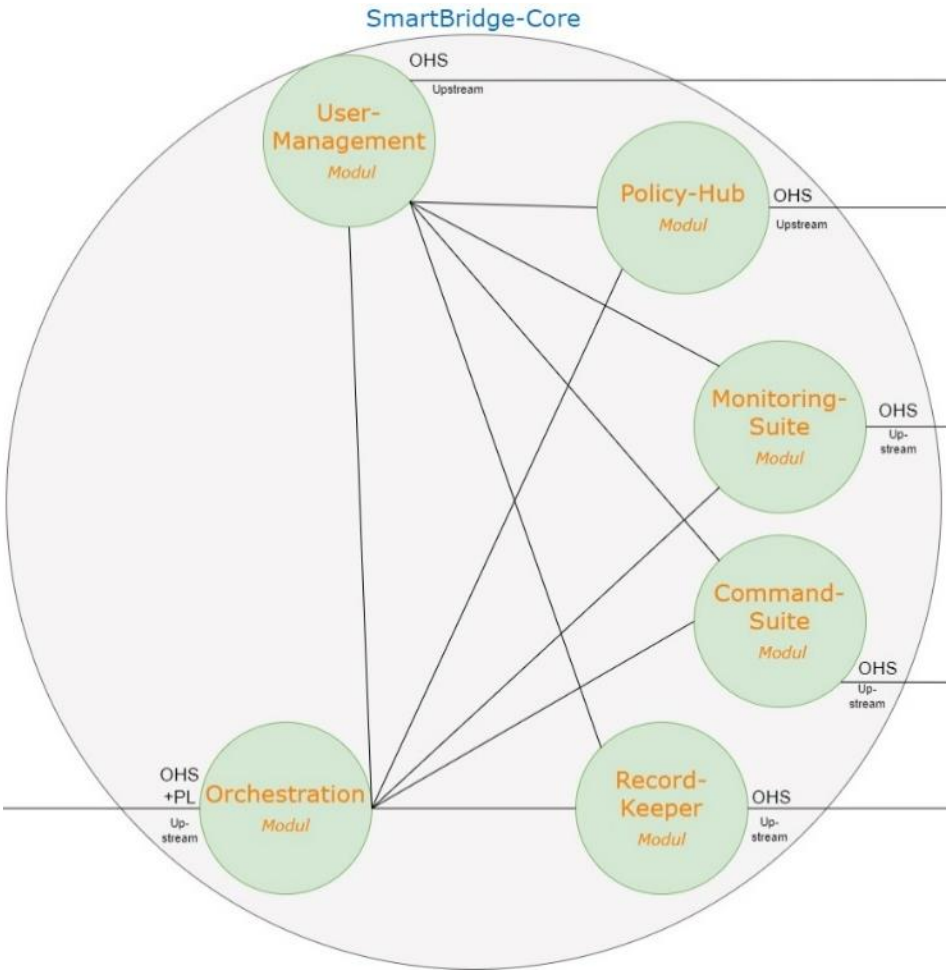


Abbildung 19: Middleware Aussenansicht

Einzig die Abhängigkeit zwischen dem Orchestration-Modul und dem in den Konnektoren befindlichen SmartBridge-Modul wurde im Unterabschnitt 4.1.3.1 - SmartBridge-Modul – Orchestration-Modul behandelt.

4.1.4.1.1. User-Management / Sonos- / Shelly- / KNX- / Mock-Connector – Orchestration

Wie auch innerhalb der Konnektoren ist die Kommunikation zwischen den Modulen innerhalb der Middleware eventbasiert. So werden die von den Konnektoren eintretenden Nachrichten entpackt und neu in gemeinsam definierten Repräsentationen als Event veröffentlicht. Jedes der Module kann darauf reagieren, sofern es dies möchte.

4.1.4.2. Abhängigkeiten zwischen Middleware-internen Modulen

Parallel zu den vorangegangenen Unterabschnitten, welche sich mit den nach aussen gerichtet Abhängigkeiten befasst haben, befassen sich nun folgende Unterabschnitte mit den Abhängigkeiten innerhalb der Middleware.

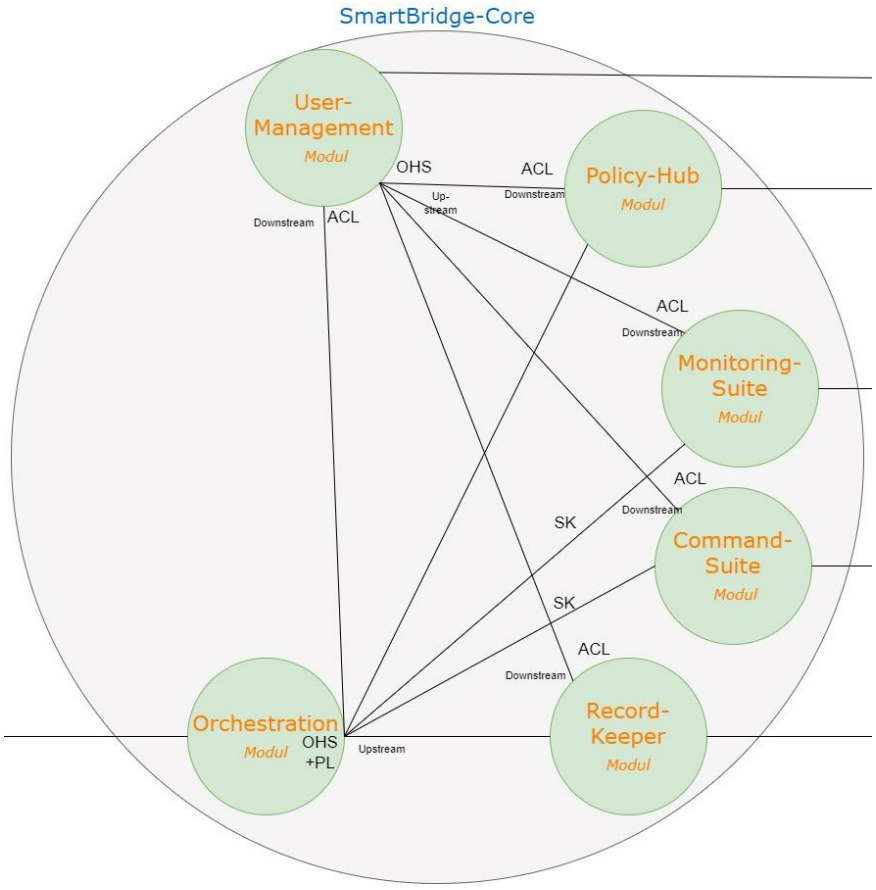


Abbildung 20: Middleware Innenansicht

4.1.4.2.1. User-Management / Policy- / Monitoring- / Command- / Record-Keeper – Demo Frontend

Die Middleware besteht ausfolgenden Funktions-Modulen:

- Policy-Hub
- Monitoring-Suite
- Command-Suite
- Record-Keeper

sowie dem Grund-Modul User-Management und dem Support-Modul Orchestration.

Diese Funktionsmodule sowie das Modul User-Management verfügen jeweils über eine eigene RESTful-http Web API, worüber die Kommunikation nach aussen gewährleistet ist. Ob die User-Interfaces die Repräsentation der Daten von den Modulen direkt übernehmen oder mittels ACL-Mappen, ist ihnen überlassen.

4.1.4.3. User-Management / Policy-Hub / Monitoring-Suite / Command-Suite / Record-Keeper

Die Hoheit über die allgemeinen Daten liegt aktuell beim Modul User-Management. Daher können diese allgemeinen User-, Connector-, Device-Attribute nur auf der API des User-Managements angepasst werden und werden darauf hin an die anderen Module, in Form eines Events, übermittelt.

Obwohl andere Module nicht alle Attribute übernehmen müssen und auch eigene Attribute auf deren Repräsentation der oben genannten Objekte führen können, können aktuell nur das User-Management-Modul und das Orchestration-Modul die anderen über Änderungen informieren. Beim Applikation-Design wurde hierbei darauf geachtet, dass durch wenige Anpassungen innerhalb der Module auch Änderungen auf den allgemeinen Attributen von den anderen Funktions-Modulen veröffentlicht werden können.

Weitere Informationen diesbezüglich sind dem ADR 9.1.11 - Daten-Hoheit der allgemeinen Daten-Attribute zu entnehmen.

4.1.4.4. Orchestration - User-Management / Policy- / Monitoring- / Command- / Record-Keeper

Da das Modul «Orchestration» für die Kommunikation zwischen den Middleware-Modulen und den Konnektoren zuständig ist, werden die anderen Module innerhalb der Middleware, durch dieses Modul über Neuerung und Änderung informiert. Dies geschieht in der Grund-Datenrepräsentation aus dem Grund, dass nicht alle Module alle Repräsentationen der anderen Module kennen müssen, um die Update-Daten interpretieren zu können.

4.1.4.5. Callback – System-APIs

Die Aufgabe des Callback-Moduls besteht darin, den Fremd-Systemen, welche einen Endpoint benötigen welcher mittels URL zugänglich ist, diesen anzubieten.

Die Interpretation der Daten geschieht dann jedoch auf dem Konnektor, obwohl der Eintritt in die Middleware-Umgebung aus Sicht des Datenflusses im Modul Callback ist. Aus logischer Sichtweise befindet sich der Eintritt in die Umgebung noch immer auf dem Konnektor. *Das Callback-Modul wurde im Rahmen dieser Arbeit auf die konzeptionelle Ebene beschränkt.*

Weitere Informationen diesbezüglich sind im ADR 9.1.9 - Sonos-Konnektor innerhalb desselben Netzwerkes wie die Sonos-Speaker beschrieben.

4.2. Messages

Allgemein stellt sich die Herausforderung, dass der Konnektor und die Middleware eine unterschiedliche Software-Version besitzen können. Dies umfasst auch die Möglichkeit verschiedener Felder.

Dennoch war es bei der Konzeption wichtig sicherzustellen, dass folgende Anforderungen umgesetzt werden:

- *Rückwärtskompatibilität*
  Es soll nicht dazu kommen, dass ein Konnektor aufgrund seiner Software-Version nicht mehr mit der Middleware kommunizieren kann. Die Konsequenz diesbezüglich ist jedoch, dass gewisse Funktionalitäten nur neuere Versionen unterstützen.

- *Anpassbarkeit der Informationen*
  Neue Informationen sollen zwischen dem Konnektor und der Middleware ausgetauscht werden können, damit auf neue allgemeine Anforderungen schnell reagiert werden kann.

Es ist eine Tatsache, dass diese Anforderungen an die Nachrichtenübermittlung ein gewisses Mass an Komplexität mit sich bringen. Trotzdem ist dies eine aktive Überlegung, die getroffen wurde zugunsten der in den Unterabschnitten 1.6.1.2 - Brauchbarkeit (usable) und 1.6.1.3 - Flexibilität (flexibel) beschriebenen Qualitätszielen.

Message-Version
Basierend auf den oben genannten Anforderungen enthält jede Message, welche zwischen dem Konnektor und der Middleware ausgetauscht wird, eine Versions-Nummer. Zu jeder Versionsnummer und Typ ist festgelegt, welche Informationen enthalten sind. So kann sichergestellt werden, dass jede Nachricht, welche versendet wird, korrekt interpretiert werden kann.

Message-Identifikator
Nebst der Versionsnummer haben auch alle ausgetauschten Nachrichten einen eindeutigen Identifikator. Dieser soll verhindern, dass dieselbe Message mehrmals verarbeitet wird.

4.2.1. Message-Themen

Um die Nachrichten nach deren Inhalten filtern zu können, ohne den Payload auspacken zu müssen, wurden verschiedene Nachrichten-Themen eingeführt. Auf diese wird in diesem Unterabschnitt genauer eingegangen.

4.2.1.1. Loging

Die Loging-Informationen von dezentralen Konnektoren können mittels Loging-Nachricht an die zentralisierte Middleware-Instanz übermittelt werden. Dies kann sehr hilfreich sein, um das Monitoring bezüglich der korrekten Funktionstüchtigkeit des Gesamtsystems und somit das Nutzer-Erlebnis zu überwachen oder auch zur Eingrenzung einer Fehlerursache. Die Idee dahinter ist, dass das Log-Level, welches auf dem Konnektor hinterlegt ist, mittels Message von der Middleware aus angepasst sowie die Übermittlung in die Cloud ein- und ausgeschaltet werden kann.

4.2.1.2. Error

Auch wenn ein Endbenutzer keine Log-Nachrichten in die Cloud übermitteln möchte, so ist es für den stabilen Betrieb und das Nutzererlebnis wichtig, dass die Meldungen, welche bei einem abgefangenen System-Absturz erstellt wurden, an die Middleware übermittelt werden.

4.2.1.3. Data Distribution

Der Endbenutzer muss gewisse Informationen dem Konnektor beim Aufsetzen mitteilen. Diese variieren stark vom angebundenen Fremd-System. Bei gewissen Fremd-Systemen kann der Konnektor die Informationen aus dem Fremd-System auslesen. Doch diese Informationen sind ebenfalls für die Middleware von Interesse und werden der Middleware mittels Data-Distribution-Message mitgeteilt. Aktuell werden diese Nachrichten-Formate nur von den Konnektoren an die Middleware gesendet. Jedoch wäre es aus Sicht der Nachricht möglich, diese an die Konnektoren zu senden, sobald sich der Use-Case diesbezüglich ergibt.

4.2.1.4. Control-Command

Alle Nachrichten, welche Informationen enthalten bei denen ein Steuer-Befehl von der Middleware aus auf die Fremd-System-Domäne übermittelt wird, werden mit dem Thema Control-Command markiert. Woher der Input für den Steuer-Befehl kommt, spielt hierbei keine Rolle.

Die Flussrichtung hierbei ist nur von der Middleware-Instanz in Richtung des Konnektors möglich.

4.2.1.5. Event-Message

Diejenigen Nachrichten, welche vom Konnektor an die Middleware-Instanz in der Cloud versendet werden und auf einem Event in der System-Domäne basieren, werden mit dem Thema Event-Message markiert.

Hierbei ist nur die Flussrichtung vom Konnektor zur Middleware-Instanz möglich.

4.2.1.6. Settings

Um Während der Laufzeit variable Einstellungen an den Konnektoren und den Modulen vornehmen zu können, werden Nachrichten mit dem Thema Settings versendet. Dieses Thema geht Hand in Hand mit der Anforderung der 1.6.1.3 - Flexibilität (flexibel).

4.2.2. Systemspezifische Nachrichten

Systemspezifische Nachrichten, sind all jene Nachrichten, welche von den Fremd-Systemen stammen und diese in Bezug auf die Abhängigkeit auch Upstream vom Konnektor liegen.

4.2.2.1. KNX-Message

Die Nachrichten, welche über den KNX-Bus gesendet werden, enthalten im Wesentlichen folgende Informationen:

- **Source**
  Physikalische Adresse vom Absender der Nachricht. *Weitere Informationen über die Physikalischen-Adressen von KNX wurden im Unterabschnitt 3.2.3.3.2 - Physikalische Adressen behandelt.*
- **Destination**
  Gruppen-Adresse, auf welche andere KNX-Geräte reagieren, sofern sie diese Adresse in ihrem internen Speicher hinterlegt haben. *Weitere Informationen über die KNX-Gruppen-Adressen wurden im Unterabschnitt 3.2.3.3.1 - Gruppen Adressen behandelt*
- **Priorität**
  Hierbei handelt es sich um die Dringlichkeit, mit welcher die KNX-Nachrichten weitergeleitet werden. (Low *[Standard-Wert]*, High, Alarm)
- **Information**
  Die Information, die der Sender dem Empfänger mitteilen möchte, wird in diesem Feld hinterlegt. Die Information ist jedoch codiert und nur mit dem entsprechenden Datentyp korrekt interpretierbar.

- **Hop-Count**
  Abnehmender Wert, der bei jedem Weiterleiten der Nachrichten von jedem Linien- und Bereichs-Koppler um eins reduziert wird und bei null nicht weitergeleitet wird. Der Startwert liegt hierbei bei sechs.
- **Physikalische Adresse des KNX-IP-Interfaces**
  Adresse des Interfaces, welches ebenfalls mit dem KNX-Bus verbunden ist.

4.2.2.2. Shelly System Message

Konnektor → Shelly-Gerät
Bei der Kommunikation von den Konnektoren hin zu den Shelly-Geräten verfügt Shelly über keine eigenen proprietären System-Nachrichten. Dies da die Kommunikation über eine RPC-API geschieht. Basierend darauf werden Request-Parameter zur genaueren Spezifikation genutzt.

Shelly-Gerät → Konnektor
Anders als bei der Kommunikation ausgehend vom Konnektor, werden bei einer Statusänderung vordefinierte Nachrichten an die Konnektoren gesendet. Dies geschieht mittels bestehender WebSocket-Verbindung. Der genaue Aufbau der Nachricht unterscheidet sich jeweils vom Gerätetyp. Es sind immer folgende Informationen vorhanden:

- IP-Adresse des Shells-Gerätes
- Eindeutige Shelly-Identifikator des Shelly-Gerätes
- ID des Kanales der die Statusänderung erfahren hat
  Dieser entspricht einem eigenständigen Gerät in den Boundet-Context der Middleware-Module
- Statusänderung
- Wie die Statusänderung ausgelöst wurde

4.2.2.3. Sonos System Message

Konnektor → Sonos-API
Ähnlich wie bei der Kommunikation vom Konnektor zu den Shelly-Geräten wird auch bei Sonos über eine API und Request-Parameter kommuniziert. Im Gegensatz zu Shelly wird bei Sonos über eine RESTful HTTP Web API kommuniziert.

Sonos-API → Konnektor
Über Statusänderungen der Sonos-Geräte informiert Sonos auf einer zu hinterlegenden «Event Callback URL». Da dies jedoch einen domainbasierten Endpunkt voraussetzt, wurde im Rahmen dieser Arbeit entschieden, auf die Kommunikation der Nachrichten vom Sonos Eco-System zu den Konnektoren zu verzichten.

4.2.3. System Nachricht

Die fremdsystemstatusbezogene Kommunikation innerhalb der Konnektoren und der Middleware, findet mittels der System-Message statt. In den folgenden Unterabschnitten werden die einzelnen Implementationen einer solchen System Nachricht erläutert.

4.2.3.1. Error-Message

Aufgrund der Tatsache, dass der Endbenutzer meist nicht unterscheidet, ob die Ursache eines fehlerhaften Verhaltens in dem vom Fremdsystem-Hersteller stammenden Konnektor oder in der Middleware liegt, sollen bei einer zukünftigen Version der Middleware alle Errors an die Middleware übertragen werden. So kann ein Monitoring über die Konnektoren aufgebaut werden.

Da die Fehlermeldungen meist schon sehr gross sind und über alle relevanten Informationen verfügen, die zum Nachvollziehen des fehlerhaften Systemzustandes notwendig ist, wurde die Error-Nachricht bewusst mit wenigen zusätzlichen Feldern versehen. Daher erhalten die Error-Messages, sofern sie ausserhalb dieser Arbeit implementiert werden, zusätzlich zu der Version und dem Identifikator noch die Exception selbst.

4.2.3.2. Third-Party-System-Message

Alle Statusänderungen an den Fremd-Systemen oder Anfragen, um diese herbeizuführen werden innerhalb der Konnektoren und der Middleware als Third-Party-System-Messages verarbeitet. Daher verfügen sie neben denselben Feldern aller Nachrichten-Typen auch über ein Objekt, das die Statusänderung angefragt hat, ein Ziel respektive ein Gerät, an dem die Statusänderung vollzogen werden soll, sowie einen Enum, welches die Statusänderung selbst repräsentiert.

Sowohl das Objekt, welches die Statusänderung anfragt als auch das Gerät, auf welchem diese vollzogen werden soll, werden durch ihren eindeutigen Identifikator repräsentiert.

Die Ausprägung der Statusänderung ist abhängig von dessen Datentyp. So ist sichergestellt, dass kein gerätefremder Status angefragt werden kann.

Die Third-Party-System-Messages werden beim Übermitteln an die Middleware mit dem Message-Thema 4.2.1.5 - Event-Message oder beim Übermitteln an den Konnektor mit dem Message-Thema 4.2.1.4 - Control-Command, versehen.

4.2.3.3. Traceability-Message

Die Tracability-Messages wurden dafür vorgesehen, um den Systemzustand der Konnektor-Module zu überwachen. Dabei wurde bewusst zwischen Error-Messages und Traceability-Messages unterschieden. So sollte der Benutzer wählen können, ob er die Tracability-Messages an die Middleware übermitteln möchte oder nicht. Wenn er sich jedoch dagegen entscheiden sollte, so würden seine Informationen auf dem User-Interface nur den beschränkten Umfang enthalten.

Eine solche Traceability-Message enthält neben der Nachrichten-Version und dem eindeutigen Identifier, über welche jeder Nachrichten-Typ verfügt, ein Textfeld, welche die Nachricht enthält. Dies wurde bewusst so gewählt, um dieselben Meldungen nutzen zu können, die auch in den konnektor-internen Logs genutzt werden. Zudem sind eher kürzere Nachrichten mit weniger Feldern passender, da es im Vergleich zu anderen Nachrichten-Typen viele Nachrichten dieses Typs geben wird. *Im Umfang dieser Arbeit wurden die Umsetzung dieses Nachrichten-Typs auf die konzeptionelle Arbeit beschränkt.*

Um den Weg der Nachrichten nachvollziehen zu können, würde man bei einer Umsetzung an allen Stellen im Code, an welchen die Nachrichten verpackt, entpackt, umgewandelt oder zusätzliche Informationen eingeholt werden, eine neue Nachricht erstellen.

4.2.3.4. Distribution-Message

Auf den drei Haupt-Datenobjekten sind diejenigen Informationen vorhanden, welche mehrere Module der Middleware interessieren oder von zentralster Bedeutung sind. Jedes Modul verfügt jedoch über eine eigene Repräsentation dieser Daten. Diese müssen jeweils nicht alle Informationen der Datenaustausch-Objekte besitzen und können auch über mehr Informationen verfügen. Zudem kann es sein, dass die Hierarchien der Daten von der des Datenaustausch-Objekts abweichen.

Der Objekt-Identifikator dient dazu, die Objekte eindeutig identifizieren zu können. Dieser referenziert ein Objekt während der gesamten Zeit über alle Systeme hinweg, was von den Konnektoren über die Middleware bis hin zum Frontend reicht.

4.2.4. Messaging Message

Bei der Wahl des Protokolls standen die folgenden von ActiveMQ unterstützten Protokolle zur Auswahl:

- *Core-Protokoll*
  Das Core-Protokoll wird von ActiveMQ Artemis als Standard verwendet und überträgt die Daten als JSON-Objekte. Der Fokus bei der Entwicklung dieses Protokolls lag auf der Leichtgewichtigkeit und einer niedrigen Latenz.

- *AMQP*
  Das AMQP ist eines der bekanntesten und verbeitetsten Messaging-Protokolle, welches den Daten-Austausch verschiedener Plattformen unterstützt.

- *MQTT*
  Die MQTT ist ein public-subscribe basiertes Protokoll, welches aufgrund der hohen Latenz und der geringen Anforderungen an die Bandbreite vorwiegend von IoT-Anwendungen eingesetzt wird.

Der entsprechende ADR zur formellen Entscheidung ist im Unterabschnitt 9.1.19 - Wahl des ActiveMQ Artemis Protokolls dokumentiert

Informationen bezüglich des genauen Nachrichtenflusses sind im Abschnitt 6.1 Messages genauer beschrieben.

4.2.4.1. Message-Version

Um die Middleware unabhängig von den Konnektoren updatebar gestalten zu können, wurde eine Versionierung der Nachricht eingefügt. *Weitere Informationen diesbezüglich stehen im ADR 9.1.18 - Versionierung der Messaging-Nachrichten.*

4.2.4.2. Message-ID

Jede Nachricht, die versendet wird, besitzt einen eindeutigen Identifikator. Hierbei handelt es sich um eine UUID.

4.2.4.3. Topic

Da es sich in der Nachricht um verschiedenste Arten von Informationen handeln kann, beschreibt dieses Feld, um welche Art von Nachricht es sich handelt. Hierbei gibt es aktuell folgende Ausprägungen:

- **Logging**
  Wenn der Konnektor log-bezogene Informationen an die Middleware übertragen möchte, so kann er dies mit diesem Nachrichten-Flag der Middleware übermitteln.
  Hintergründe über die Nachrichten zu diesem Topic sind im Unterabschnitt 4.2.1.1 - Loging ausgeführt.

- **Error**
  Alle Errors, welche bei den Konnektoren auftreten, müssen an die Middleware übermittelt werden. Dies soll der Middleware einen Überblick über den Systemzustand geben.
  Hintergründe über die Nachrichten zu diesem Topic sind im Unterabschnitt 4.2.1.2 - Error ausgeführt.

- **Data Distribution**
  Gewisse Informationen, die der Endbenutzer auf dem Konnektor erfasst, sind auch für die Middleware von Interesse. So übermittelt der Konnektor diese mithilfe dieses Nachrichten-Topic der Middleware.
  *Hintergründe über die Nachrichten zu diesem Topic sind im Unterabschnitt 4.2.1.3 - D*ata Distribution *ausgeführt.*

- **Control command**
  Dieser Nachrichten-Typ wird dann verwendet, wenn die Middleware dem Konnektor eine Anweisung zum Schalten eines Ausganges auf einem Gerät geben möchte. Solche Nachrichten werden normalerweise immer von der Middleware an die Konnektoren gesendet.
  Hintergründe über die Nachrichten zu diesem Topic sind im Unterabschnitt 4.2.1.4 Control-Command ausgeführt.

- **Event Message**
  Im Gegensatz zu den zuvor beschriebenen «Control command», werden die Event Messages normalerweise von den Konnektoren an die Middleware versendet. Dies um der Middleware mitzuteilen, dass auf dem Fremd-System ein Event stattgefunden hat.
  Hintergründe über die Nachrichten zu diesem Topic sind im Unterabschnitt 4.2.1.5 - Event-Message ausgeführt.

- **Settings**
  Der Endbenutzer soll in der Zukunft auch die Möglichkeit erhalten, konnektor-spezifische Anpassungen auf dem Frontend der Middleware vornehmen zu können. Daher wurde dieser Topic-Typ erfasst.
  Hintergründe über die Nachrichten zu diesem Topic sind im Unterabschnitt 4.2.1.6 - Settings ausgeführt.

4.2.4.4. System-Message

Die eigentliche Nachricht ist im Feld «System Message» verpackt. Aktuell kann eine «SmartBridgeJmsMessage» nur eine System-Message enthalten. Um die Übertragung der Nachrichten jedoch effizienter gestalten zu können sind die System-Messages bereits vorbereitet, um sie eindeutig unterscheiden zu können und somit mehrere System Messages in derselben «SmartBridgeJmsMessage» einzubetten.

In der Message-Version 1.0 sind folgende System Message-Formate implementiert:

- *Error-Message*
    - Message-Version
    - Message-ID
    - Exception

  Weitere Informationen zu diesem Nachrichten-Typ sind dem Unterabschnitt 4.2.3.1 - Error-Message zu entnehmen.

- *System-Message*
    - Message-Version
    - Message-ID
    - Source
    - Target
    - Content

  Weitere Informationen zu diesem Nachrichten-Typ sind dem Unterabschnitt 4.2.3.2 - Third-Party-System-Message zu entnehmen.

- *Traceability-Message*
    - Message-Version
    - Message-ID
    - Message

  Weitere Informationen zu diesem Nachrichten-Typ sind dem Unterabschnitt 4.2.3.3 - Traceability-Message zu entnehmen.

4.2.4.5. Message creation Timestamp

Da es bei der Übermittlung mittels Messaging-Plattform zu Verzögerungen kommen kann, wurde dieses Feld hinzugefügt, um eine allfällige zeitliche Priorisierung der Nachrichten vornehmen zu können.

5. Building Block View

Der Abschnitt Building Block View richtet sich an die Hersteller von IoT- und Smart Home-Geräten und an User-Interface Entwickler. Dieser soll ihnen die konkrete technische Umsetzung der Arbeit näherbringen.

Dieses Kapitel Baueinsicht geht auf die folgenden Software-Komponenten ein und beschreibt dessen technische Umsetzung sowie die Struktur und dessen Besonderheiten:

- Konnektoren
- Middleware
- Demo-User-Interface

5.1. Gesamt-Übersicht

Die Grundstruktur innerhalb der Spring-Modulith-Module ist jeweils gleich aufgebaut. Dies ist sowohl bei den Konnektoren als auch bei der Middleware der Fall.

Sowohl die Middleware als auch die Konnektoren wurden als Modularen Monolithen implementiert. Bei diesem Architektur-Design werden die Module als eigenständig lauffähige Programme designt.

5.1.1. Allgemeiner Aufbau der Module

Innerhalb der Module ähnelt sich der Aufbau der Ordner und Dateien sowohl in der Struktur als auch in der Benennung. Dies soll die Zeit der Einarbeitung in ein neues Modul auf ein Minimum reduzieren.

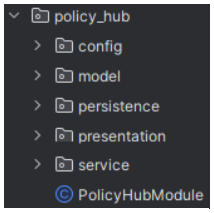


Abbildung 21: Beispiel-struktur eines Moduls

5.1.1.1. Config

Im Order Config» liegen alle für dieses Modul relevanten Einstellungen. Darunter auch alle Informationen, welche Spring JPA und die Datenbank benötigt, um die Daten zu persistieren und in die Java-Objekte umzuwandeln.

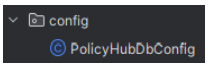


Abbildung 22: Beispielstruktur des Ordners "config"

5.1.1.2. Model

Alle modulspezifischen Daten-Objekte werden innerhalb dieser Dateistruktur definiert. Darunter können sowohl Daten-Austausch-Objekte (dto's) oder auch Entitäts-Objekte fallen.

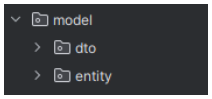


Abbildung 23: Beispielstruktur des Ordners "model"

5.1.1.3. DTO

Beim Erstellen von neuen Daten-Instanzen über eine RESTful http Web API, werden bei gewissen Endpunkten spezifische Daten-Austausch-Objekte erwartet. Dies sind Repräsentationen der neu zu erstellenden Daten-Instanz mit reduziertem Informationsumfang.

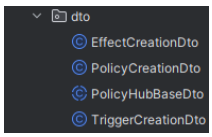


Abbildung 24: Beispielstruktur des Ordners "dto"

5.1.1.3.1. Entity

Wenn ein Daten-Objekt in die Datenbank persistiert werden soll, so muss es sich innerhalb dieses Ordners befinden um von Spring erkannt zu werden. Die Einstellungen, innerhalb welcher Ordner-Struktur Spring nach den Entitäten des entsprechenden Modulith-Modules sucht, ist im DB-Konfigurationsdatei des Modules definiert. Diese befindet sich im Konfigurations-Ordner des jeweiligen Moduls.

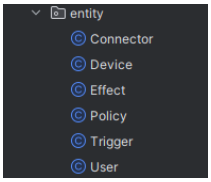


Abbildung 25: Beispielstruktur des Ordners "entity"

5.1.1.4. Persistence

Alle Repository-Klassen, welche als Interface zischen der Applikation und der Datenbank dienen, sind im Persistence-Ordner abgelegt. Diese Interfaces bieten eine Abstraktion der Datenbankoperationen an. Zudem werden alle für die Entität spezifischen Datenbank-Abfragen innerhalb dieser Repository-Klassen definiert.

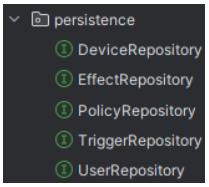


Abbildung 26: Beispielstruktur des Ordners "persistence"

5.1.1.5. Presentation

Die Controller der Endpoints für den Zugriff auf die Kontroller via API, wurden im Orner «Presentation» abgelegt. Sie stellen die API zur Verfügung, enthalten jedoch keinerlei Logik. Sämtliche Logik, welche spezifisch den Endpoint betreffen, befinden sind in den Controller-Services. Diese befinden sich innerhalb der gleichnamigen Ordner unter der Haupt-Ordnergruppe «Service».

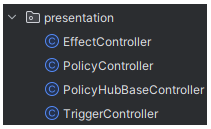


Abbildung 27: Beispielstruktur des Ordners "presentation"

5.1.1.6. Service

Die Service-Klassen verfügen jeweils über die spezifischen Logik-Informationen zur Operation der Objekte. Darunter fallen sowohl die Controller-Services, die Core-Services als auch die modul-spezifischen Services.

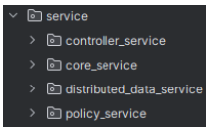


Abbildung 28: Beispielstruktur des Ordners "service"

5.1.1.6.1. Controller-Services

Sowohl das Zusammentragen der Informationen als auch das Koordinieren der auszuführenden Funktions-Logik eines RESTful http Web API-Endpoints gehören zu den Aufgaben der Controller-Services. So hat jeder Controller eine eigene Kontroller-Service-Klasse. Diese kann dann auch auf mehrere Core-Services oder modul-spezifischen Services zugreifen.

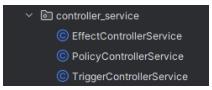


Abbildung 29: Beispielstruktur des Ordners "controller_service"

5.1.1.6.2. Core-Service

Jede Entität verfügt über ihre eigene Core-Service-Klasse. Diese ist die einzige Anlaufstelle für alle anderen Klassen auf die Entitäten. Nur diese Klasse darf auf das Repository zugreifen. Dies hat den Vorteil, dass eine Klasse verantwortlich ist für den Zugriff und Verteilung der Entität.

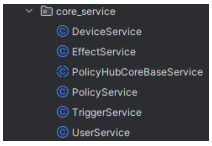


Abbildung 30: Beispielstruktur des Ordners "core_service"

5.1.1.7. Distributed-Data-Service

Um die Grund-Datentypen wie Benutze, Konnektor und Gerät zwischen den Modulen und deren Datenbanken synchroneren zu könne, wurden die im Ordner «Distributed-Data-Service» befindlichen Klassen eingeführt. *Weitere Informationen zu den genauen Synchronisationen in Bezug auf die jeweiligen Module sind im Abschnitt 5.4 - Middleware dokumentiert.*

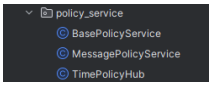


Abbildung 31: Beispielstruktur des Ordners "distributed_data_service"

5.1.1.8. System-Service

Die System-Services umfassen alle modulinternen Service-Klassen.

Sofern sich das Modul im Konnektor befindet, regelt diese Klasse alle Zugriffe auf das entsprechende Fremd-System.

Befindet sich das Modul hingegen innerhalb der Middleware, behandeln diese Klassen die Domänen-Logik des Moduls.

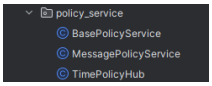


Abbildung 32: Beispielstruktur des Ordners "system_service"

5.2. Test-Aufbau

Um die verschiedenen Elemente des Ecosystems, wie die Konnektoren, die Middleware oder das Demo User-Interface testen zu können, wurde eine Test-Umgebung angefertigt.

5.2.1. Netzwerk

Die weit verbreiteten Geräte und dessen Hersteller unterstützen meist verschiedene Protokolle zur Kommunikation. Doch praktisch alle unterstützen aufgrund der weiten Verbreitung das Internet Protocol. Daher werde ich im Rahmen dieser Arbeit ein solides Fundament mittels IP-Netzwerk aufbauen und die einzelnen Smart Home- und IoT-Lösungen darauf aufbauen.

Übersicht

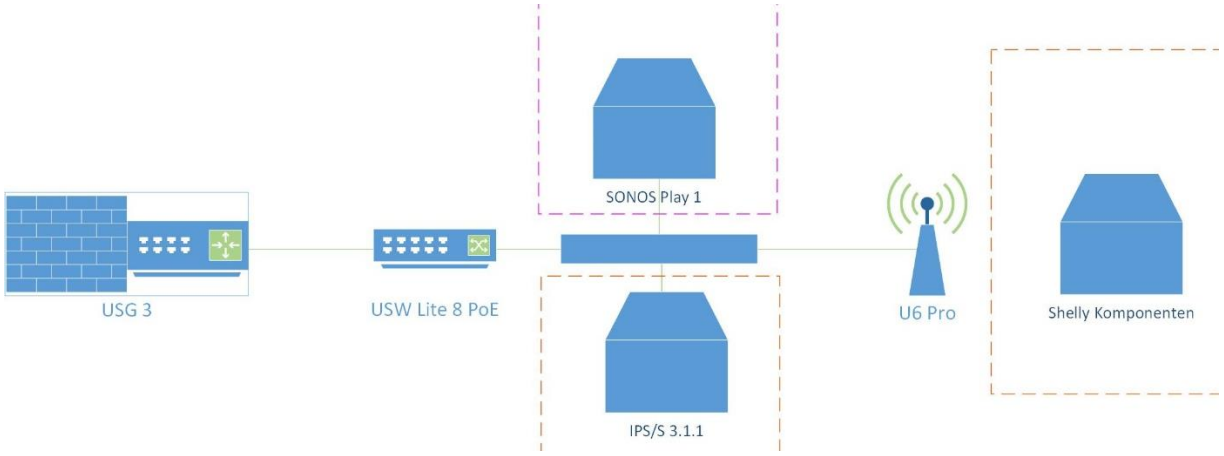


Abbildung 33: Netzwerkübersicht des Testaufbaus

Router
Modell: Ubiquiti UniFi Security Gateway
Hersteller: Ubiquiti Networks
Version: 4.4.57

Access Point
Modell: Access Point U6 Pro
Hersteller: Ubiquiti Networks
Version: 6.6.65

Switch
Modell: Ubiquiti 8-Port Lite PoE Switch
Hersteller: Ubiquiti Networks
Version: 6.6.61

5.2.2. KNX-Umgebung

IP-Schnitstelle
Modell: KNX-IP-Schnittstelle IPS/S 3.1.1
Hersteller: ABB
Version: 3.1.1

Spannungsversorgung
Modell: SV/S30.640.3.1
Hersteller: ABB

Schaltaktor
Modell: Schaltaktor 4fach 16 A
Hersteller: Feller by Schneider Electric
Version: 36304-4 REG

Taster
Modell: EDIZIOdue colore KNX-Taster RGB
Hersteller: Feller by Schneider Electric
Version: 4704-2-B.FM.L.P.67

5.2.3. Shelly-Umgebung

Modell: Shelly Plus i4
Hersteller: Allterco Robotics Ltd

Modell: Shelly Pro 4PM
Hersteller: Allterco Robotics Ltd

Modell: Shelly Plus 2PM
Hersteller: Allterco Robotics Ltd

5.2.4. Sonos-Umgebung

Lautsprecher
Modell: Sonos PLAY:1
Hersteller: Sonos Inc.

5.2.5. IP-Konzept

Host-Adresse: 192.168.118.1

Subnetz: 192.168.118.0/24 (255.255.255.0) => 254 Nutzbare Hosts

Netzwerk-Range: 192.168.118.2 – 20

- Core-Geräte: 192.168.118.2 – 9
- Access-Points: 192.168.118.10 – 19

DHCP-Range: 192.168.118.20 – 149

IoT-Range: 192.168.118.150 – 254

- KNX-Range 192.168.118.150 – 179
- Shelly-Range 192.168.118.180 – 229
- Sonos-Range: 192.168.118.230 – 249

Shelly Komponenten:

- «PlusI4» 192.168.118.180
- «PlusI4» 192.168.118.181
- «Pro4PM» 192.168.118.182
- «Plus2PM» 192.168.118.183

5.2.6. ETS-Konfiguration

Die ETS oder Engineering Tool Software ist jene Software, mit der ein KNX-Gebäudeleitsystem konfiguriert werden kann. In dieser wird für jedes Gebäude ein eigenes Projekt angelegt, welches einen digitalen Zwilling der Gebäude darstellt.

5.3. Konnektoren

Die Konnektoren sind diejenigen, eigenständig lauffähigen Software-Komponenten des Ecosystems, welche für die Kommunikation mit den Fremd-Systemen zuständig sind.

5.3.1. Architektur-Übersicht

Bei der Entwicklung der Konnektoren wurde, ebenso wie bei der Middleware selbst, das Spring-Framework verwendet. Dabei kam das Modulith-Framework zum Einsatz. Die Überlegung hinter der Entscheidung auch einen modularen Monolithen zu bauen war, sich die Möglichkeit offen zu halten zu einem späteren Zeitpunkt die system-basierten Module herauszulösen und dann jeweils in demselben Netzwerk nur ein Verbindungsmodul zur Middleware zu benötigen. *Den genauen ADR zu dieser Entscheidung ist im Unterabschnitt 9.1.12 - Wahl der Anzahl Verbindungs-Module zwischen Konnektoren und Middleware beschrieben.*

Daher wurden die Konnektoren so designt, dass sie jeweils über ein systemnahes Modul und eines zur Kommunikation mit der Middleware verfügen. Das systemnahe Modul nimmt das Mapping der Bounded-Context vor und kann somit auch als ACL angesehen werden. Dieses wurde nach dem Fremd-System benannt und unterscheidet sich stark bezüglich der Funktionsweise und dem Umfang zwischen den Fremd-Systemen. Bei jenem Modul, welches die Kommunikation zwischen dem Konnektor und der Middleware verarbeitet wurde darauf geachtet, dass keine Konnektor-spezifischen Anpassungen notwendig sind. Dies aus dem Grund, dass zu einem späteren Zeitpunkt dieses Modul herausgelöst und als Dependency hinzugefügt werden kann.

Beim Design und der Entwicklung wurde aktiv darauf geachtet und mittels Tests auch sichergestellt, dass es keine Abhängigkeiten zwischen den Modulen gibt. Hiervon ausgenommen sind speziell annotierte Objekte im Common-Modul.

Sicherer Informationsaustausch zwischen Konnektor und Middleware

Die Kommunikation zwischen dem Konnektor und der Middleware wurde aktuell so implementiert, dass keine Verschlüsselung der Daten stattfindet. Vor einer allfälligen Veröffentlichung müsste man eine Verschlüsslung mittels symmetrischem AES-Verfahren sowohl seitens der Konnektoren als auch auf Seiten der Middleware einbetten.

Hierzu müssten noch folgende Arbeiten gemacht werden:

- Die beim erstmaligen Start des Konnektors ausgetauschten Informationen müssten um die RSA-Public-Key's erweitert werden.

- Der konnektor-interne Key-Value-Speicher Mechanismus müsste mindestens wie folgt angepasst werden:
    - Datenbank-File verschlüsseln
    - passwort-geschützter Zugriff sicherstellen

- Das Konnektor-Modul «SmartBridge» und das Middleware-Modul «Orchestration» muss vor dem Senden der Nachricht diese mit dem AES-Schlüssel verschlüsseln, signieren und beim Entpacken nebst dem Verschlüsseln die Signatur prüfen.

Den formellen ADR bezüglich der Entscheidung die Datenübertragung aktuell nicht zu verschlüsseln ist im Unterabschnitt 9.1.15 - Sicherheit der Datenübertragung zischen Konnektor und Middleware erläutert.

5.3.1.1. Aufbau der Konnektoren

Jeder Konnektor besteht aus den folgenden drei Modulen:

- Common-Modul
- system-basiertes Modul
- SmartBridge

5.3.1.1.1. Common-Modul

Das Common-Modul ist aus funktionaler Sicht kein vollständiges Modul. Es wird aktuell dazu verwendet Objekte, welche in mehreren Modulen verwendet werden, zentral abzulegen. Zu einem späteren Zeitpunkt, wenn sich die Strukturen dieser Objekte herauskristallisiert haben, können diese Objekte als Dependency zur Verfügung gestellt werden. Weiter stellt dieses Modul einen Mechanismus zum Persistieren von Key-Values zur Verfügung. *Wie dieser Mechanismus genau funktioniert, ist im Unterabschnitt 5.3.1.3.1 - MapDB weiter ausgeführt.*

Der Endbenutzer hat die Möglichkeit die folgenden Einstellungen manuell anzupassen:

- ActiveMQ Artemis URL
- Konnektor ID
- Benutzer-ID
- SmartBridge-Core Domain
- Pfad zur Initialisierung des Konnektors auf der Middleware

Bezüglich der User-ID ist zu erwähnen, dass diese durch die Anmeldung mit den Zugangsdaten der Middleware abgelöst werden sollte bevor der Konnektor eingesetzt wird. Hierzu ist ein Frontend nützlich jedoch nicht Umfang dieser Arbeit.

Key-Value-API

Um einem allfälligen Frontend des Konnektors eine Möglichkeit zu bieten die benutzerspezifischen Einstellungen anzupassen, bietet der Shelly-Konnektor entsprechende Endpoints. Es wurden folgende HTTP-Methoden umgesetzt:

- *GET*
- *POST*
- *PUT*
- *DELETE*

Für die http-Methoden GET und PUT ist jeweils der Key des Key-Value-Paares notwendig.

Global-Settings-Service

Innerhalb des Shelly-Konnektors stehen die Values allen Modulen und den darin enthaltenen Klassen über den Key-Value-Service zur Verfügung. Dieser Service stellt dieselben Methoden zur Verfügung wie die anderen Service-Klassen. Sie entspricht einem Controller-Service da es durchaus möglich ist, dass für andere zentrale Einstellungen noch weitere Persistenz-Lösungen hinzukommen. Der Zugriff ausserhalb dieses Moduls soll auch über diesen Service geschehen.

5.3.1.1.2. System-basierte Module

Das system-basierte Modul ist für das Mapping der Kontexte verantwortlich. Da dies auch die Anreicherung des Informationsumfangs bedeuten kann, verfügen diese Module über eine eigene Relationale Datenbank. *Die Anforderungen und welche Datenbank genau eingesetzt wurden steht im Unterabschnitt 5.3.1.2 - System-DB geschrieben.*

Der Aufbau dieser Module ist strukturell immer gleich, um sich schneller in den Konnektoren auszukennen. *Genauere Informationen über den Aufbau der Module sind im Unterabschnitt 5.1.1 - Allgemeiner Aufbau der Module beschrieben.*

Frontend
Im Rahmen dieser Arbeit wurde die Eingabe der servicebasierten Daten auf ein RESTful http Web API beschränkt. Dies stellt einen Kompromiss zwischen dem Verzicht einer direkten Interaktion des Endbenutzers mit dem Konnektor und einem vollwertigen Frontend dar. Die Anforderung für eine direkte Interaktion des Endbenutzers tauchte erst während der Arbeit durch den Umstand auf, dass zusätzliche Informationen über System-Geräte benötigt werden, um eine eindeutige Identifikation der betroffenen Geräte vornehmen zu können.

Kontext
Das system-basierte Modul und deren System-Datenbank ist immer im Bounded-Context des Fremd-Systems implementiert. Dies aus dem Grund, dass die Einstellungen und Angaben die der Endbenutzer oder dessen Fremdsystem-Spezialist, wie zum Beispiel ein KNX-Integrator, im Kontext des Fremd-Systems vornehmen kann. Das Mapping des Kontextes wird dann vom Systembasierten Modul vorgenommen, während dem Versenden und Empfangen der Nachrichten an oder von der Middleware.

5.3.1.1.3. SmartBridge Modul
Das Modul «SmartBridge» ist für die gesamte Kommunikation zwischen den Konnektor und der Middleware zuständig. Hierbei werden die Nachrichten, welche von der Middleware verpackt und anschliessend von ActiveMQ Artemis übermittelt werden, entpackt und anhand des Topics innerhalb des Konnektors veröffentlicht. Die Veröffentlichung der Informationen wird durch den von Spring-Modulith zur Verfügung gestellten Application-Event-Publisher vorgenommen. Dieser läuft in einem eigenen Thread und übergibt allen Methoden mit der Annotation «@EventListener» die Nachricht, sofern sie im Übergabe-Parameter der Methode über denselben oder einen Untertyp der Nachricht verfügen.

5.3.1.1.4. Fremd-System-Dependency
Um den Implementationsaufwand, während der POC-Phase effizient zu gestalten, wird bei den Fremd-Systemen, bei denen es bereits Java-Projekte gibt, welche die Kommunikation mit dem Fremdsystemen übernehmen, auf diese Projekte mittels Dependency zugegriffen.

Beim KNX-Konnektor wurde das «Calimero-Projekt» und beim Sonos-Konnektor die «Sonos API client for Java» genutzt.

5.3.1.2. System-DB
Jeder Konnektor verfügt über eine eigene Relationale Datenbank. Diese ist dem system-basierten Modul vorbehalten. Ihr Nutzen liegt im Persistieren derjenigen Daten, welche für das Mapping des Kontexts vom Fremd-System und dem der Middleware benötigt wird.

Anforderungen an System-DB
Der Konnektor wurde darauf ausgelegt, dass er nach dem Start ohne weiteren Neustart durchgehend laufen soll. Um jedoch auch nach einem allfälligen Stromausfall noch alle persistierten Daten zur Verfügung zu haben, bestand die Anforderung die Daten dauerhaft persistieren zu können. Gleichzeitig sollte die Datenbank leichtgewichtig sein und den Verwaltungsaufwand sowie die damit verbundenen Anforderungen an die Kenntnisse zu verringern. Um die ACID-Eigenschaften und die Möglichkeit zu komplexen SQL-Abfragen sicherstellen zu können, war die Verwendung von Relationalen Datenbanken von Vorteil.

Entscheid für H2 als System-Datenbank
Basierend auf den oben erwähnten Anforderungen wurden die Datenbanken SQLite und H2 genauer evaluiert. Aufgrund der Möglichkeit von Stored Procedures und der Unterstützung von Geospatial Data fiel die Entscheidung auf H2. Dies, um für die Implementation weiterer Fremd-Systeme offen zu sein, ohne die Datenbank später wechseln zu müssen. *Der formale ADR bezüglich dieser Entscheidung ist im Unterabschnitt 9.1.13 - Evaluation der System-Datenbank im Konnektor beschrieben.*

Die H2-Datenbank wird nicht wie meist üblich als In-Memory-Datenbank, sondern als persistente File-Datenbank betrieben. Dies stellt zum einen die dauerhafte Persistenz sicher (auch nach einem Stromausfall) und bietet zum anderen die Möglichkeit regelmässige Sicherungen der Datenbank zu erstellen. Zudem sind zur Sicherung der Datenbank keine datenbank-spezifischen Kenntnisse erforderlich.

Physikalischer Ort der System-Datenbank
Um die Latenz so gering wie möglich zu halten, befindet sich die Datenbank auf derselben Umgebung wie die Applikation. Zudem ist der User für dessen Sicherheit verantwortlich. Die Alternative hierzu wäre der Betrieb in der Cloud. Dies würde den Abgleich der System-Komponenten vereinfachen doch überwogen die Argumente für den lokalen Betrieb. *Weitere Informationen diesbezüglich sind dem ADR 9.1.14 - Physischer Standort der System-Datenbank zu entnehmen.*

Zugriffs-Sicherheit versus Usability
Der Zugriff auf die in der System-Datenbank befindlichen Daten wird aktuell noch nicht mit einem Passwort geschützt. Dies wurde aktiv zugunsten der Usability entschieden, da die Konnektoren sich innerhalb des Netzwerkes des Endbenutzers befinden und sich darin keine weiteren Informationen als diejenigen die meist durch Scannen des Netzwerkes verfügbar sind befinden.

Dennoch steht die Vergabe eines Passworts auf der Liste der Aufgaben, die ausserhalb und nach Abschluss dieser Arbeit erledigt werden müssen.

5.3.1.3. Key-Value-Speicher Mechanismus
Maven bietet zwar die Möglichkeit Konfigurations-Parameter in einer Datei namens «application.propperties» zu hinterlegen. Diese stehen dann den Entwicklern innerhalb der gesamten Applikation zur Verfügung. Beim Start der kompilierten Applikation können diese Konfigurations-Parameter überschrieben werden. Doch benötigt es diesbezüglich gewisse Kenntnisse, wie eine kompilierte Applikation beim Start Übergabe-Parameter mitgegeben werden können und wie diese zu formatieren sind. Um dies zu vereinfachen, wurden in dieser Datei nur Konfigurations-Parameter hinterlegt, bei denen es sehr unwahrscheinlich ist, dass der Endbenutzer diese ändern muss.

Dennoch wurde ein Mechanismus benötigt, wie applikationsweit einfach auf Konfigurations-Parameter zugegriffen werden kann. Zusätzlich musste die Voraussetzung geschaffen werden dem User die Möglichkeit zu geben Einfluss auf diese zu nehmen. Dies wurde mittels Wrappers um eine Datenbank sichergestellt. Dies wurde mit der 5.3.1.4 - Global-Settings Datenbank vom Typ 5.3.1.3.1 - MapDB umgesetzt.

5.3.1.3.1. MapDB
Bei der Auswahl der Konfigurationsparameter-Datenbank war es wichtig, dass die Konfigurations-Parameter auch bei einem Neustart der Applikation vorhanden sind sowie dass sie über eine geringe Latenz verfügen.

Bewusster Entscheid gegen H2 als Konfigurations-Parameter-Datenbank
Da der Konnektor bereits über eine Datenbank verfügt wäre der Einsatz der H2-Datenbank für den beschriebenen Einsatz-Bereich naheliegend. Die Art wie die Konfigurations-Parameter in der «application.propperties»-Datei gespeichert werden legt den Einsatz eines Key-Value-Stores näher als den Einsatz einer Relationalen Datenbank. Bei der Wahl der Konfigurations-Parameter-Datenbank fiel der Entscheid auf eine Datenbank namens MapDB.
Weitere Informationen dazu, warum MapDB und nicht H2 eingesetzt wird sind im ADR 9.1.17 - MapDB anstelle von H2 für das Persistieren von Konfigurations-Parametern beschrieben.

Was ist MapDB?
Die MapDB ist in Java geschrieben, unterstützt die ACID-Eigenschaften, kann sowohl als In-Memory-Variante als auch als persistente Variante betrieben werden und wurde mit dem Fokus auf eine geringe Latenz entwickelt. Im Gegensatz zu Relationalen Datenbanken, die ihre Daten in einer Tabelle speichern, persistiert MapDB ihre Daten in der vom Anwender vorgegebenen Collection-Datenstruktur.

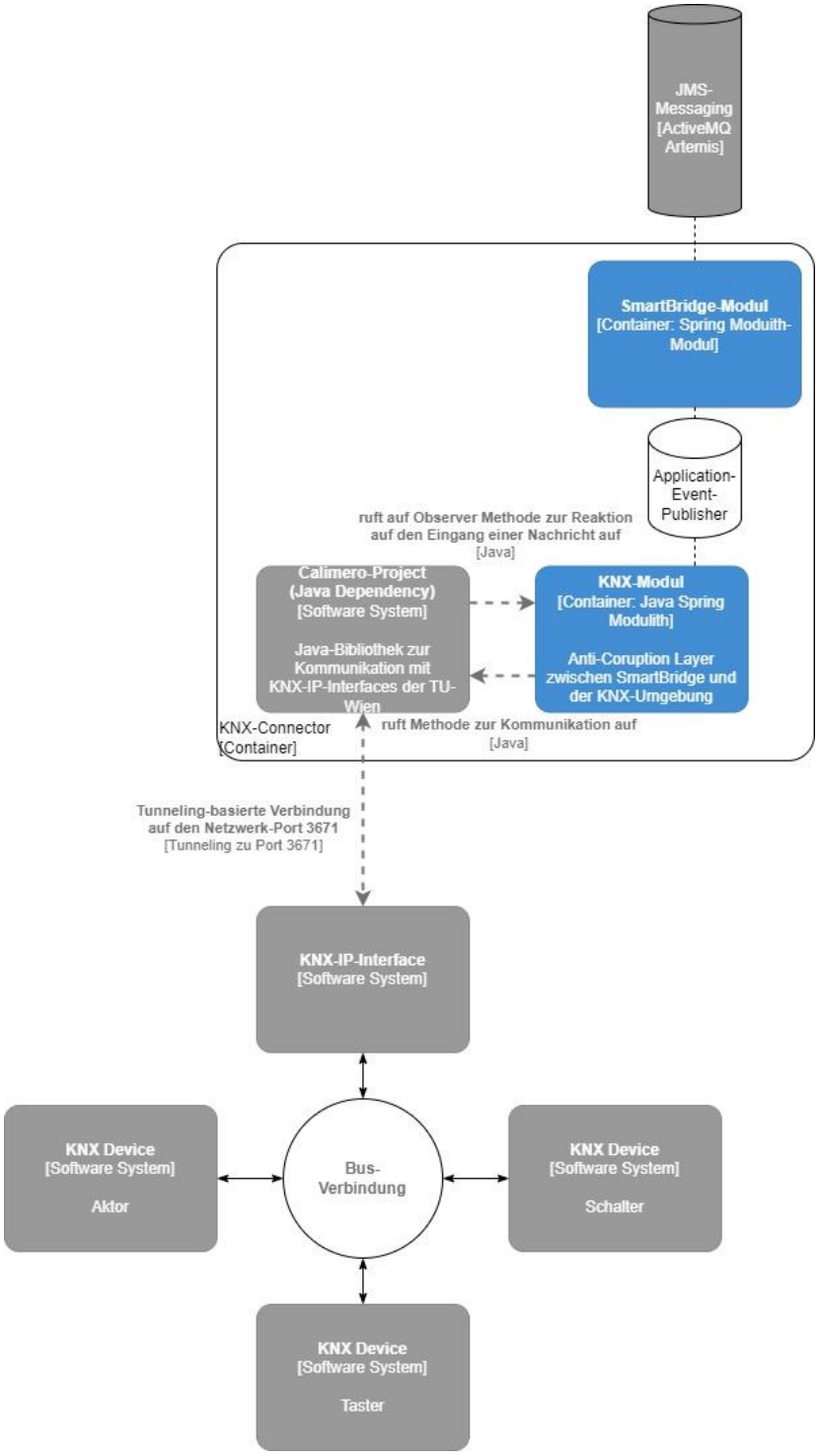
5.3.1.4. Global-Settings Datenbank
Beim Persistieren der globalen einstelluungen wird sowohl der Key als auch der Value als String gespeichert. Um sowohl Performanz, Atomizität als auch Thread-Sicherheit garantieren zu können verwendet MapDB eine Concurrent-Map.

5.3.2. KNX-Konnektor

In den folgenden Unterabschnitten wird die Architektur des KNX-Konnektors thematisiert.

5.3.2.1. Container Diagramm

Nachstehend wird die Grobübersicht des KNX-Konnektors in Bezug auf dessen Architektur erläutert.



Speziell beim KNX-Konnektor ist, dass die Kommunikation mit dem KNX-Interface nicht durch das KNX-Modul sondern über das Calimero-Projekt erfolgt.

Abbildung 34: Container Diagramm des KNX-Knnektors

5.3.2.2. Komponenten Diagramm

Weiterführende Informationen über die Architektur des KNX-Konnektors sind folgendem Diagramm zu entnehmen:

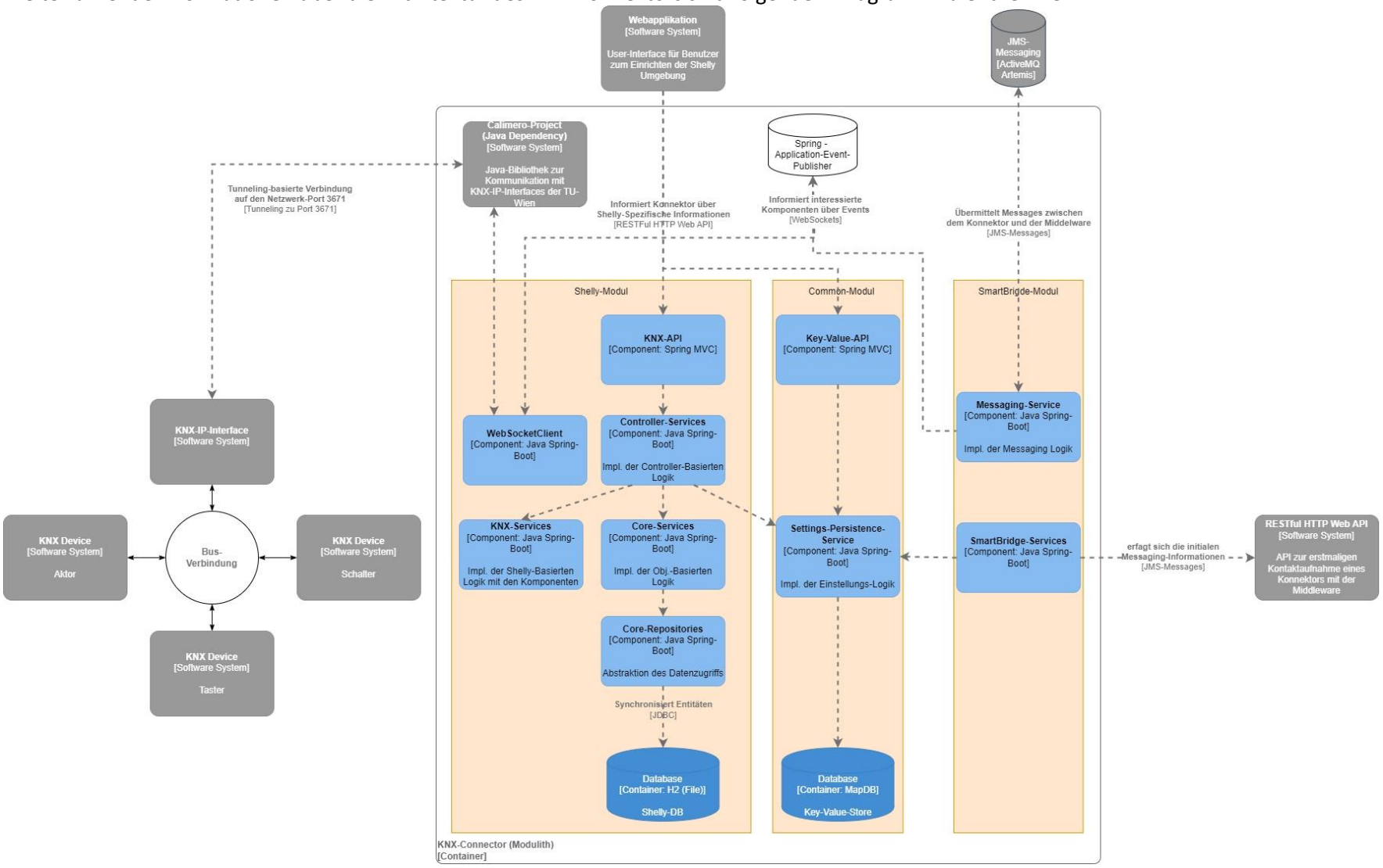


Abbildung 35: Komponenten Diagramm des KNX-Konnektors

5.3.2.2.1. Calimero

Die Java-Library mit den Projekt-Namen Calimero ist für die Kommunikation zwischen dem KNX-Interface und dem Konnektor verantwortlich. Die Nachrichten befinden sich bei der Übergabe an das KNX-Modul noch immer in der Repräsentation des Bounded-Context von KNX.

Die Kommunikation mit dem KNX-IP-Interface, welches sowohl an das IP-Netzwerk als auch an den KNX-Bus angeschlossen ist, geschieht mittels Tunnelings über den von der IANA speziell für die KNX-Kommunikation reservierten Port 3671.

Das Calimero-Projekt wird als externe Bibliothek Dependency in das Projekt eingebunden.

5.3.2.2.2. KNX-Modul

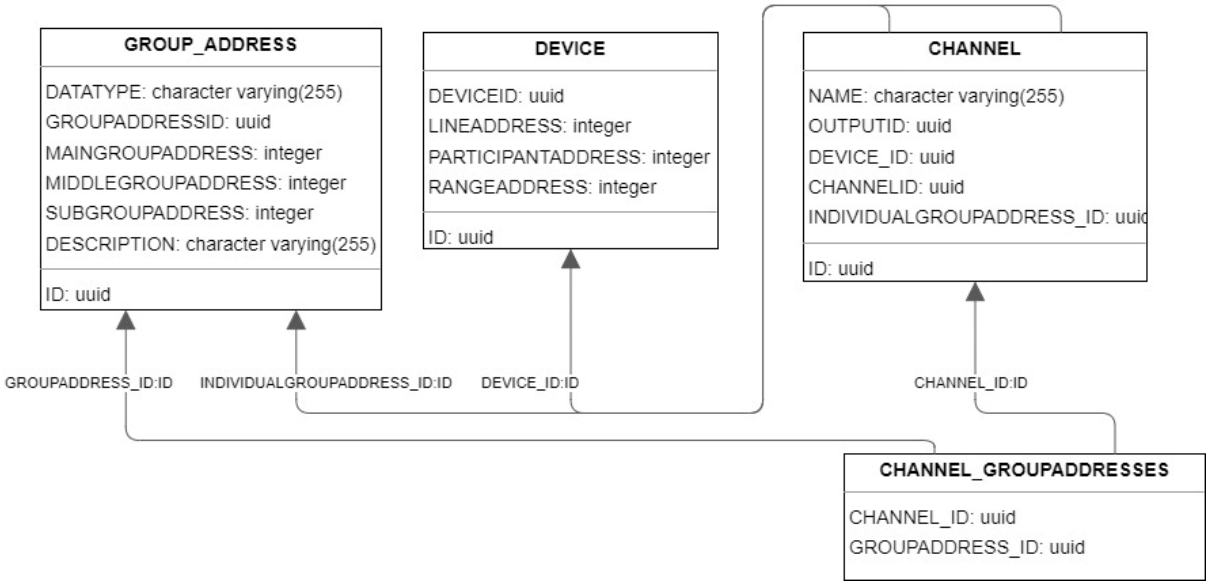
Wie auch die anderen System-Module hat das KNX-Modul die Hauptaufgabe, die Nachricht in den neuen Kontext umzuwandeln und den Informationsgehalt entsprechend zu erweitern.

Hierbei ist die Herausforderung, dass der Empfänger einer KNX-Nachricht keine Geräteinstanz, sondern eine Adresse ist, auf welche die Geräte reagieren, sofern sie diese bei sich im internen Speicher hinterlegt haben. Daher muss bei jeder Anweisungs-Nachricht, welche vom KNX-System übermittelt wird, in der internen System-Datenbank nach den Geräten gesucht werden, bei denen diese Adresse hinterlegt ist, um die Empfänger zu evaluieren. So kann es sein, dass eine KNX-Nachricht keine oder mehrere SmartBridge-Nachrichten auslöst.

Eine Konsequenz der KNX-System-Architektur ist es, dass alle Geräte mit samt ihren Gruppen-Adressen, (jene Adressen, welche in den internen Speichern der Geräte stehen und auf die sie reagieren) manuell in die Datenbank des Konnektors eingepflegt werden. Beim Design des KNX-Standards vor mehr als 20 Jahren war das Auslesen der Gruppenadressen aus den Geräten keine Anforderung und wird daher nicht unterstützt. Diese Informationen können nur der entsprechenden KNX-Projekt-Datei entnommen werden.

Eine Möglichkeit die Usability zu verbessern wäre, das automatisierte Parsen der in XML formatierten KNX-Projekt-Datei und die gewonnenen Informationen anschliessend in die System-Datenbank zu persistieren.

5.3.2.3. KNX-Datenbank



5.3.3. Shelly-Konnektor

In den folgenden Unterabschnitten wird die Architektur des Shelly-Konnektors thematisiert.

5.3.3.1. Container Diagramm

Nachstehend wird die Grobübersicht des Shelly-Konnektors in Bezug auf dessen Architektur dargestellt:

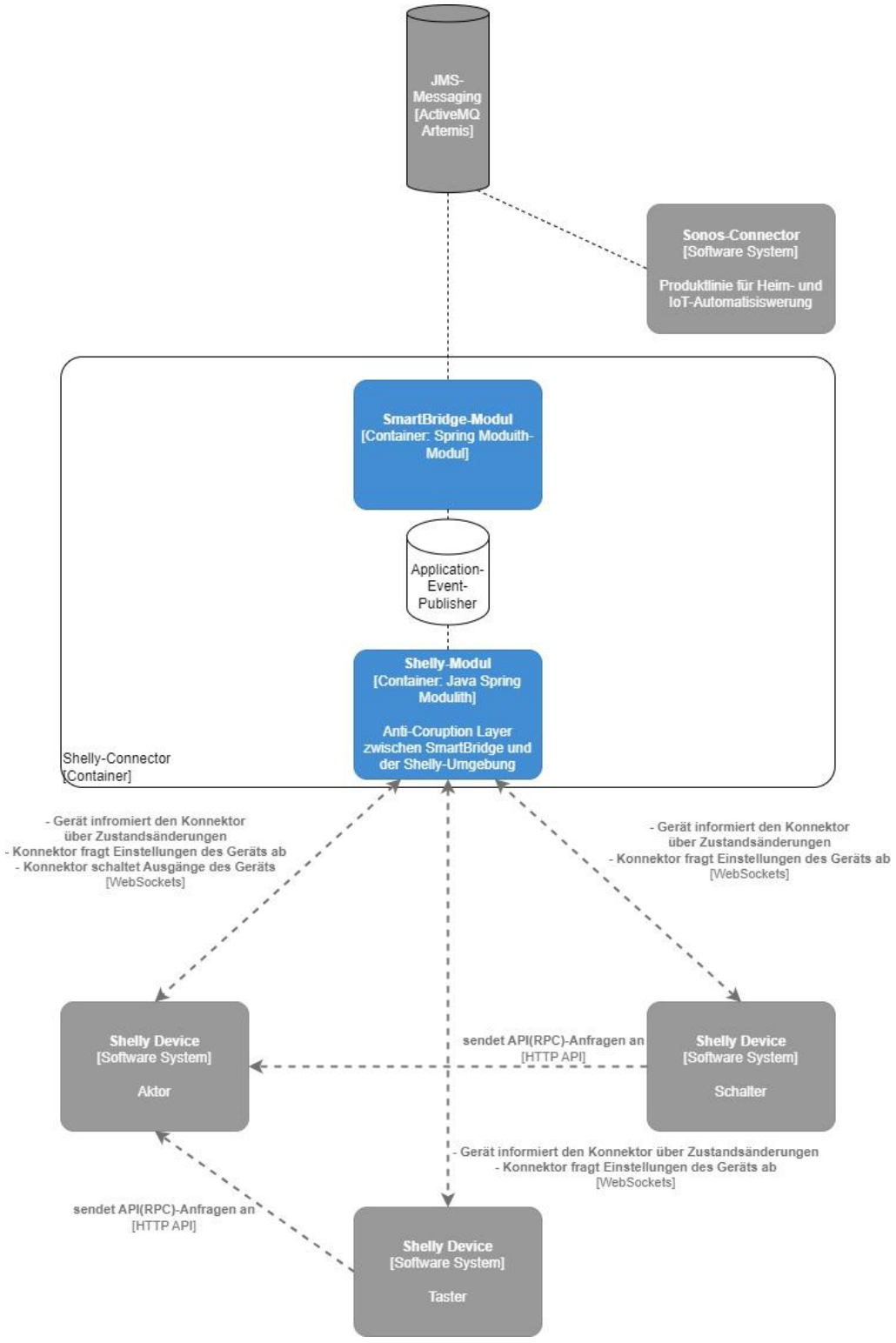


Abbildung 36: Container Diagramm des Shelly-Konnektors

Wie bereits im Unterabschnitt 5.3.1.1 - Aufbau der Konnektoren ausführlich erläutert, ist das Modul SmartBridge für die Kommunikation mit der Middleware über ActiveMQ Artemis zuständig und das Modul Shelly für das Mapping der Kontexte.

Shelley-Modul
Die Besonderheit des Shelly-Moduls ist es, dass für jedes Shelly-Gerät, welches angebunden ist, eine eigene WebSocket-Verbindung aufrechtgehalten wird. Darüber werden alle Informationen, abgesehen von der Suche nach Geräten des Auto-Discover-Endpoints, ausgetauscht.

Sowohl das Abfragen von Informationen auf den Geräten als auch das Schalten von Ausgängen bei Aktoren geschieht über die von Shelly zur Verfügung gestellten RPC-Endpoints. Zudem informieren die Shelly-Geräte ihr gegenüber jeweils über Status-Änderungen, wenn eine WebSocket-Verbindung besteht. *Weitere Informationen bezüglich der zur Verfügung stehenden Endpoints sind im Unterabschnitt 3.2.1 - Shelly-Kontext beschrieben.*

Die Shelly-Geräte, welche auch via App oder Cloud-Applikation konfiguriert werden können, kommunizieren untereinander per Standard via HTTP API mittels RPC-Anfragen.

Application-Event-Publisher
Innerhalb des Konnektors kennen sich die beiden Module nicht. Sie veröffentlichen jeweils nur Events oder reagieren auf diejenigen Events, auf die sie sich basierend auf dem Observer-Pattern (Gamma et al., 1994, S. 293-303) registriert hatten.

5.3.3.2. Komponenten Diagramm

Weiterführende Informationen über die Architektur des Shelly-Konnektors sind folgendem Diagramm zu entnehmen.

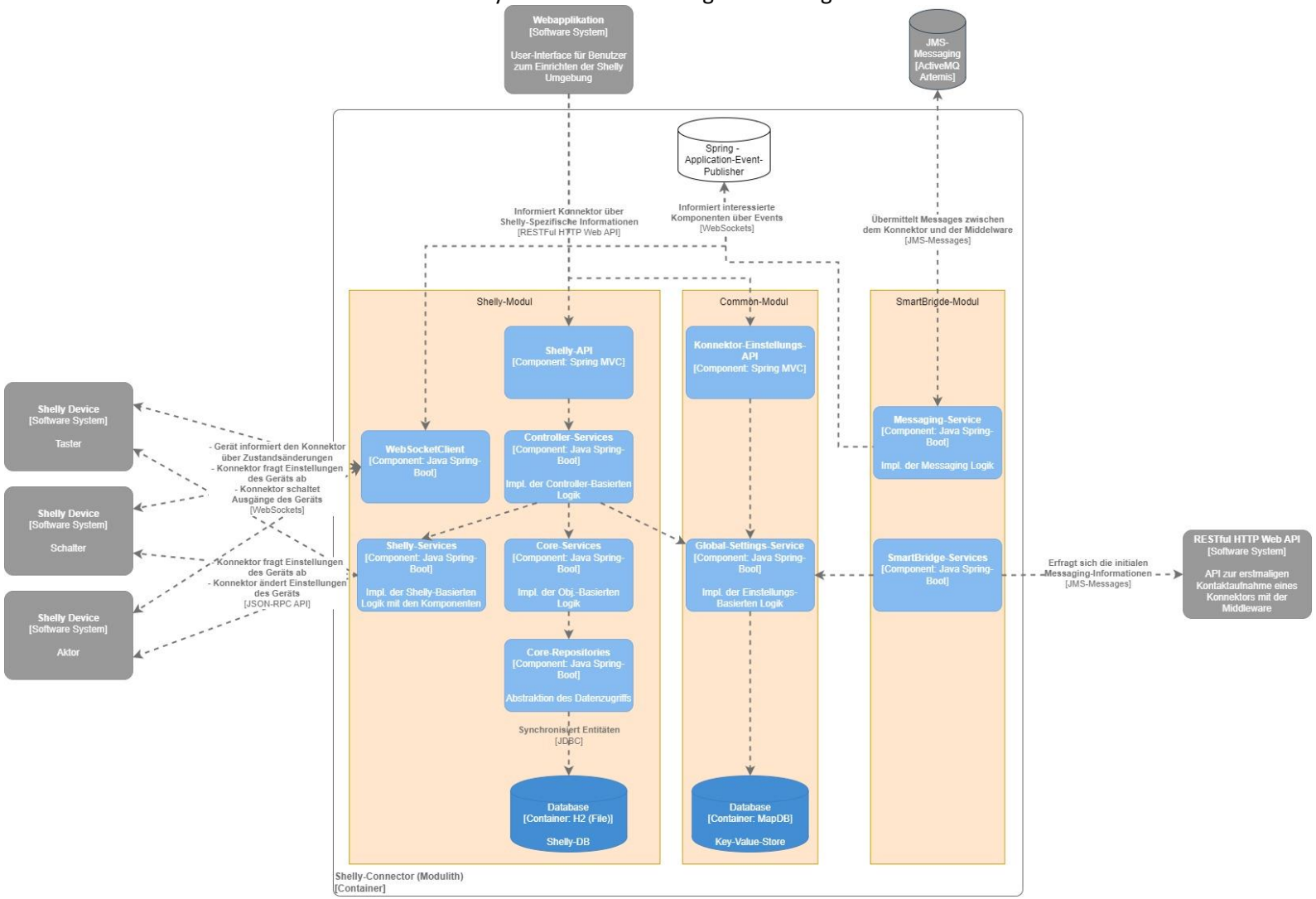


Abbildung 37: Komponenten Diagramm des Shelly-Konnektors

5.3.3.3. Shelly-Modul

Shelly-API
Alle Informationen, die der Endbenutzer dem Konnektor mitteilen möchte, kann er über die Endpunkte dieser API mitteilen. Zudem sollen in Zukunft, noch detailliertere Informationen aus dem Shelly-System ausgelesen und die Einstellungen, die auf den Shelly-Geräten möglich sind angepasst werden können. Der Umfang dieser Arbeit deckt die Umsetzung diesbezüglich nicht ab.

Die Controller haben die Aufgabe die API-Anfrage an die ControllerService-Klassen weiterzuleiten und deren Antworten als Antwort zurückzugeben.

Controller Services
In den Controller-Klassen werden alle Informationen aus den Core-Services zusammengetragen und in die erwartete Repräsentation strukturiert.

Core-Services
Aus Sicht des Shelly-Moduls sind die meisten Informationen an dem Geräte-Objekt aufgehängt. Sämtliche Logik, die nur eine Entität betrifft, werden in den Core-Service-Klassen vorgenommen.

Im Shelly-Modul gibt es Core-Service-Klassen für Devices und Hooks. Alle anderen Daten-Objekte sind Teil der zuvor genannten Entitäten.

Core-Repositories
Alle Datenbankabfragen, die den Core-Services zur Verfügung stehen, werden in den Core-Repositories definiert.
Die genaue Repräsentation der Daten sind im Unterabschnitt 5.3.3.4 - Shelly-Datenbank beschrieben.

Shelly-Services
Da Shelly seinen Anwendern die Möglichkeit bietet auf den Shelly-Geräten die Einstellungen anzupassen und den jeweiligen Status via RPC auszulesen, sind die hierzu vorbereiteten Methoden in den Shelly-Service-Klassen dafür vorgesehen.

WebSocket Clients
Jedes Gerät erhält vom Shelly-Modul einen eigenen virtuellen Thread für den eigenständigen Austausch von Informationen. Die Kommunikation hierbei geschieht mittels eigener WebSocket-Verbindung. Seitens des Shelly-Moduls wird mittels «Application-Event-Publishers» und dessen Events kommuniziert.

5.3.3.3.1. Shelly-API

Die Shelly-Konnektor-API verfügt über mehrere Endpunkte wobei beim Einrichten eines neuen Konnektors zwei davon essenziell sind.

Autodiscovery-Endpunkt
Der erste der beiden essenziellen Endpunkte fragt alle IP-Adressen, welche sich innerhalb derselben Subnetzmaske befinden ab, ob sie auf eine Shelly-Adresse antworten. Wenn dies der Fall ist, so geht der Konnektor davon aus, dass dies ein Shelly-Gerät ist und fragt die eindeutige Kennung der neu gefunden Shelly-Komponente ab. Dies und die IP-Adresse, unter welcher die Komponente gefunden wurde, wird in einer Liste als Antwort zurückgegeben.

Dieser Endpunkt soll dem Endbenutzer das Einrichten eines Shelly-Konnektors stark vereinfachen, da er nicht wissen muss, an welcher IP-Adresse sich die Geräte befinden.

Request: http://localhost:8091/api/v1/shelly/discover/autodiscovery?timeout=250

Response:


```
{
  "_links": {
    "self": {
      "href": "http://localhost:8091/api/v1/shelly/discover/autodiscovery?timeout=250"
    }
  },
  "shellypro4pm-c8f09e87e46c": [
    "192.168.118.182"
  ],
  "shellyplus2pm-c82e18071d34": [
    "192.168.118.183"
  ],
  "shellyplusi4-d4d4da3ae24c": [
    "192.168.118.180"
  ],
  "shellyplusi4-64b7080b1134": [
    "192.168.118.181"
  ]
}
```


Devices

Der zweite essenzielle Endpunkt einen Shelly-Konnektor einzurichten, ist der Endpunkt, um die neu gefundenen Geräte zu speichern. Der Beweggrund dies nicht automatisch während des Suchlaufes einzubinden war, dass der Endbenutzer nicht gezwungen wird alle Shelly-Komponenten hinzuzufügen.

Parallel zum Persistieren der hinzugefügten Shelly-Komponenten in die lokale Datenbank des Konnektors werden die einzelnen Sensor- und Schalt-Kanäle der Shelly-Komponenten in Form einer Update-Nachricht an die Middleware gesendet und dort an alle interessierten Module weiterverteilt, so dass alle betroffenen Module über die neuen Geräte (im Sinne des Bounded-Context der Middleware) informiert werden.

Request: http://localhost:8091/api/v1/shelly/devices


```
[
  {
    "ipAddress": "192.168.118.182",
    "shellyDeviceId": "shellypro4pm-c8f09e87e46c"
  },
 {
    "ipAddress": "192.168.118.183",
    "shellyDeviceId": "shellyplus2pm-c82e18071d34"
  },
 {
    "ipAddress": "192.168.118.180",
    "shellyDeviceId": "shellyplusi4-d4d4da3ae24c"
  },
 {
    "ipAddress": "192.168.118.181",
    "shellyDeviceId": "shellyplusi4-64b7080b1134"
  }
]
```


Response:


```
{
  "_links": {
    "self": {
      "href": "http://localhost:8091/api/v1/shelly/devices"
    }
  },
  "192.168.118.180": "fd83695c-8ea8-402e-b80d-b7f4a7becc82",
  "192.168.118.181": "60233618-42b4-4168-b8a2-2a650e2cda17",
  "192.168.118.182": "c4459c25-92f6-4707-ba8b-e2d81327cc0a",
  "192.168.118.183": "51623814-5c57-4326-8160-1105d96679c5"
}
```

5.3.3.4. Shelly-Datenbank

Basierend auf der Tatsache, dass die Shell-geräte nur die Status-Updates ihrer Elektrotechnischen Zustandsänderungen an die Konnektoren übermitteln, muss die logische Interpretation sowie dass damit verbundene halten des Zustands von Konnektor übernommen werden. Daher eingeht eine komplexere Datenhaltung als bei den anderen Konnektoren.

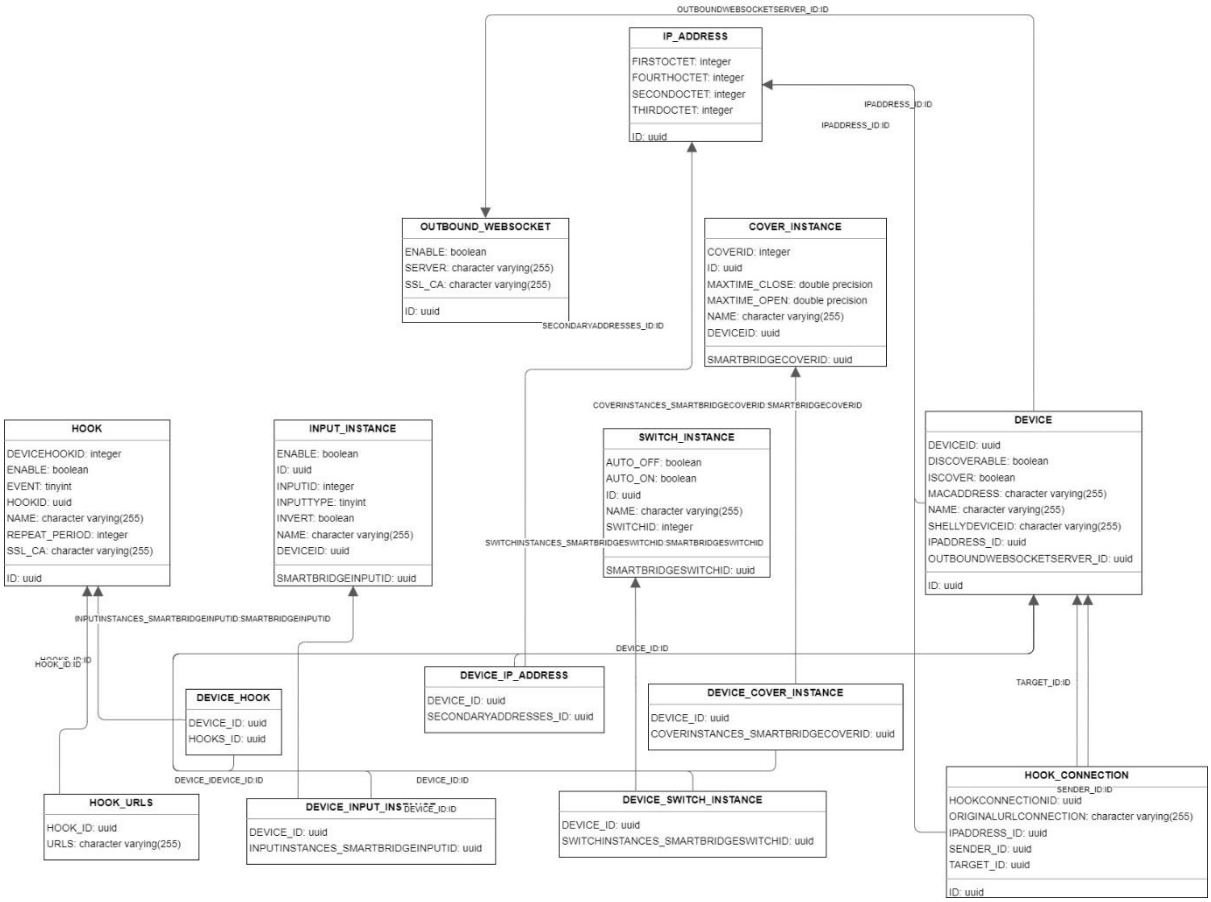


Abbildung 38: Shelly-Spezifische Datenbank des Shelly-Konnektors

5.3.4. Sonos-Konnektor

In den folgenden Unterabschnitten wird die Architektur des Sonos-Konnektors thematisiert.

5.3.4.1. Container Diagramm

Nachstehend ist die Grobübersicht des Sonos-Konnektors in Bezug auf dessen Architektur dargestellt:

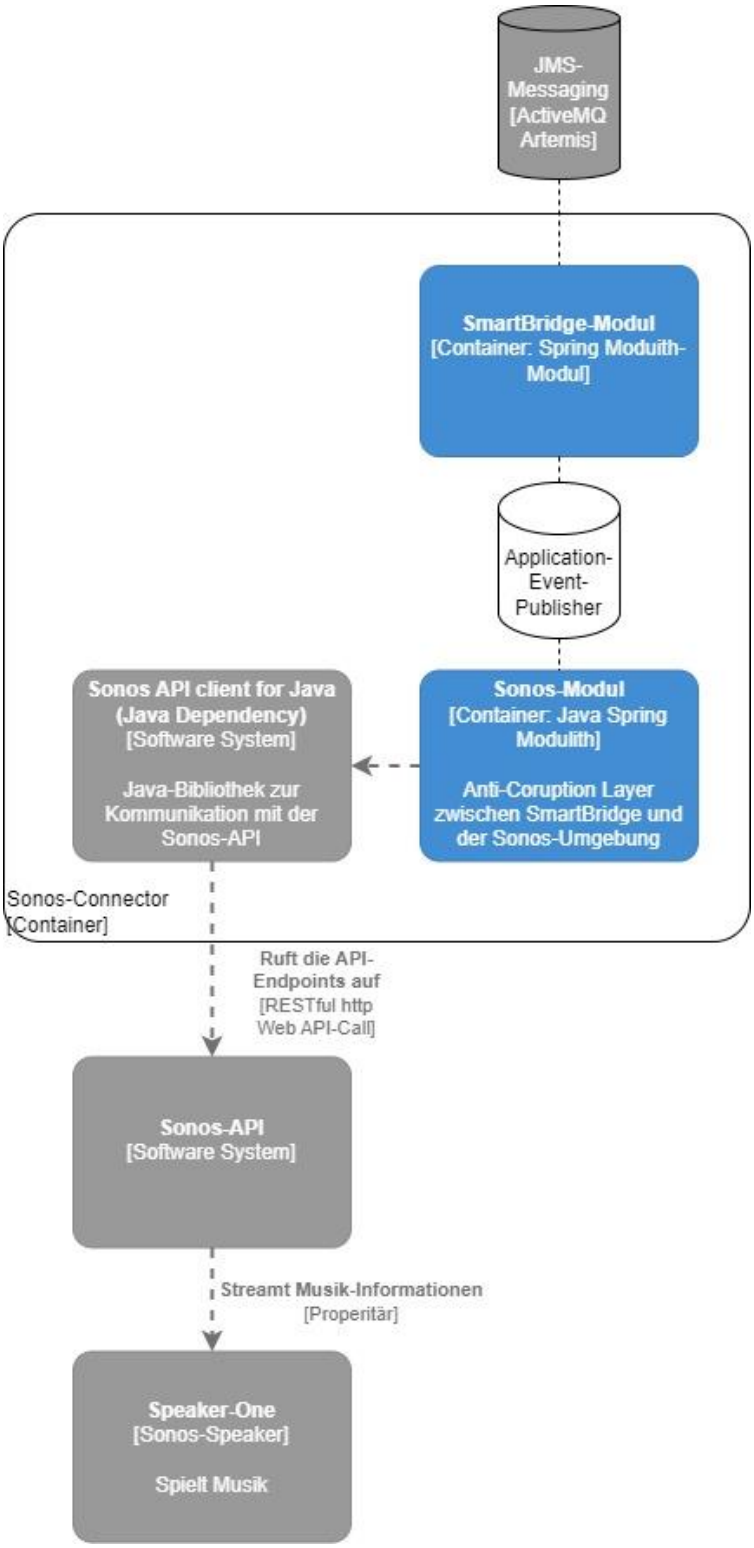


Abbildung 39: Container Diagramm des Sonos-Konnektors

Wie auch beim KNX-Konnektor, wird auch hier für die Kommunikation mit dem Fremd-System, in diesem falle die Sonos-API, eine externe Abhängigkeit verwendet.

5.3.4.2. Komponenten Diagramm

Weiterführende Informationen über die Architektur des Sonos-Konnektors sind folgendem Diagramm zu entnehmen.

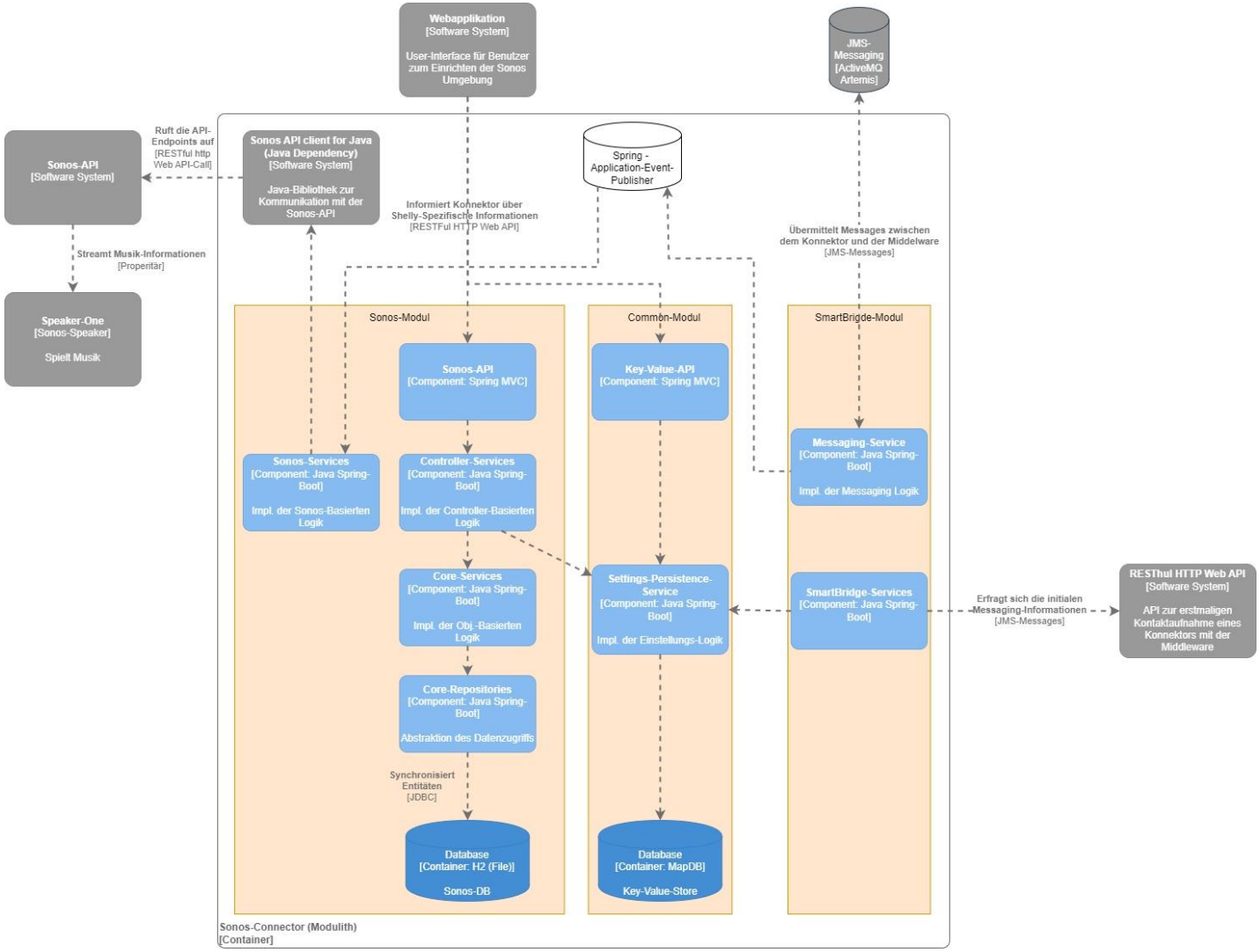


Abbildung 40: Abbildung 11: Komponenten Diagramm des Sonos-Konnektors

5.3.4.3. Sonos Modul

Das Sonos-Modul ist für das Mapping des Bounded-Context von Sonos und dem der Orchestration aus der Middleware verantwortlich. Eines der wichtigsten Mappings ist hierbei, dass im Kontext von Sonos zwischen den Sonos-Lautsprechern und den Gruppen, in welchen mehrere Lautsprecher gruppiert werden können, unterschieden wird. Im Kontext der Middleware werden nur die Sonos-Gruppen dargestellt. Dies aus dem Grund, dass sich der Funktionsumfang, welcher dem User zur Verfügung steht, davon unterscheidet, ob die Funktion auf einer Gruppe oder einem Gerät ausgeführt werden soll. Diese Abstraktionsschicht kann bei anderen Herstellern von ähnlichen Systemen anderweitig implementiert sein.

Während der Implementation musste festgestellt werden, dass sich der Identifikator, welcher eine Gruppe von Lautsprechern referenziert, ändern kann. Um nun zu verhindern, dass eine Schalt-Anweisung auf eine Gruppe referenziert, deren Identifikator bereits geändert hat, werden beim Eingang einer Schalt-Anweisung auf dem Sonos-Modul die Gruppen neu bei der API abgefragt, bevor die Schaltanweisung an die Sonos-API übermittelt wird.

Beim erstmaligen Start des Konnektors, muss der Endbenutzer den Sonos-Konnektor autorisieren auf sein Sonos-System zuzugreifen. Bei diesem Autorisierung-Prozess stellt Sonos einen Referenz-Code aus, welcher dann nach dem Start des Konnektors hinterlegt werden muss, bevor der Konnektor die Informationen des Endbenutzers auf der API von Sonos abrufen kann. Durch den Authentifizierungs-Prozess wird der Benutzer mittels HTML-File geführt.

5.3.4.4. Sonos API client for Java – Dependency

Der hinter dem GitHub-Account «nightowlengineer» stehende Entwickler James Milligan, unterhaltet einen inoffiziellen Java-Client zur Sonos-API. Diese hat er unter der MIT-Lizenz veröffentlicht und sie auf diese Art und Weise anderen Entwicklern zur Verfügung gestellt.

Der inoffizielle Java-Client stellt, basierend auf den API-Endpunkten, die Funktionalitäten als Methoden zur Verfügung. Somit konnte bei der Implementation des Sonos-Modules innerhalb des Sonos-Konnektors der Overhead der Serialisierung und des API-Calls erstellen verzichtet werden.

5.3.4.5. Common-Modul

Neben den allgemeinen Daten-Austausch-Objekten hat der Endbenutzer die Möglichkeit, den im Sonos-Modul verwendeten Autorisierung-Code von Sonos zu speichern. Ansonsten unterscheidet sich das Common-Modul nicht wesentlich zu den Implementationen in den anderen Konnektoren.

5.3.4.6. Sonos-Datenbank

In der konnektor-internen Datenbank werden nur die notwendigen Informationen gespeichert. Darunter fallen alle Token-Informationen, Infos über die Player und über deren Gruppen. Dies hauptsächlich um sicherzustellen, dass wenn eine neue Gruppe hinzugefügt wird oder die Zusammenstellung der Player in den Gruppen ändert, dies auch der Middleware mitgeteilt wird.

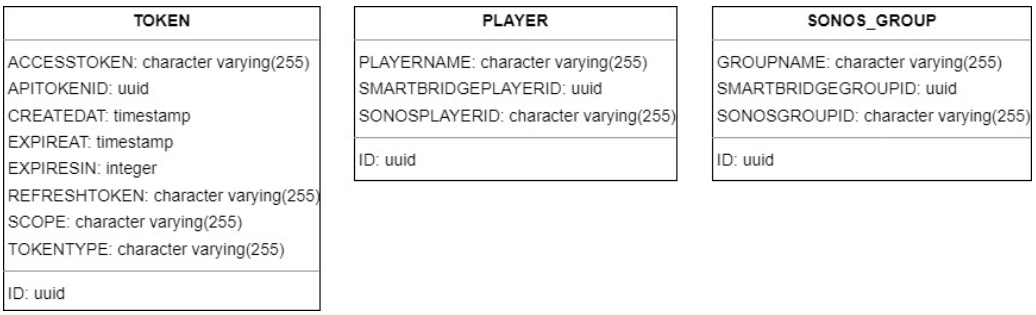


Abbildung 41: Sonos-Spezifische Datenbank des Sonos-Konnektors

5.3.4.7. Global-Settings Datenbank

Neben dem Autorisierung-Code von Sonos sind im Sonos-Konnektor keine spezifischen Einstellungen des Endbenutzers gespeichert.

5.3.4.8. Besonderheiten

Status-Updates

Änderungen auf dem Sonos-System werden aktuell nicht an die Middleware übermittelt. Dies hat den Grund, dass die Statusänderungen nicht von innerhalb des Heim-Netzwerkes des Endbenutzers verfügbar sind.

Sonos informiert Fremd-Systeme über die Änderungen an dem System des Endbenutzers über die hinterlegte Callback-URL des fremden Systems. Da sich der Sonos-Konnektor jedoch im System des Benutzers befindet, kann dieser nicht erreicht werden.

Die Lösung wäre ein Modul in der Middleware, um die Nachrichten von solchen Fremd-Systemen an den entsprechenden Konnektor weiterzuleiten. Dies wurde jedoch nicht im Rahmen dieser Arbeit umgesetzt.

Der entsprechende ADR kann im Unterabschnitt 9.1.7 - Callback-Modul nachgelesen werden.

API-Antworten

Bei einigen Anfragen an die von Sonos zur Verfügung gestellten API-Endpunkten wurden widersprüchliche Informationen zurückgegeben.

Beispielhaft wird nachstehend einer dieser Endpunkte aufgeführt:
POST https://api.sonos.com/login/v3/oauth/access
Wenn die Credentials ungültig sind, wird folgende Responce zurückgegeben:

Status Code: *400*

Body:

```
{
  "error": "server_error",
  "error_description": ""
}
```

Abbildung 42: Sonos-API Response

Widersprüchlich dabei ist, dass der Status-Code *400 - bad request* als Fehler des Clients definiert ist. Hingegen die Fehler-Meldung im Message-Body der Antwort besagt einen Fehler des Servers. Dies führt zur Verunsicherung, ob der Fehler serverseitig oder clientseitig aufgetreten ist.

5.4. Middleware

Modularten

Die Module in der Middleware lassen sich in zwei Kategorien einteilen: Funktionale Module und Support-Module. Die Aufteilung in die einzelnen Module erfolgte nach dem «Separation of concerns»-Prinzip.

5.4.1. Middleware-Architektur

Die Middleware wurde nach der Modularer-Monolith (Modulith)-Architektur entwickelt. Diese Architektur vereint die beiden Architekturen «Micro-Services» und «Monolith». Hierbei wird die Applikation in verschiedene Module unterteilt, analog zur Microservice-Architektur, jedoch werden sie in einer Instanz deployet.

Die Modulith-Architektur geht die Problematik des YAGNI-Prinzips aus der Microservice-Architektur an. Diese besagt, dass einige der umgesetzten Anforderungen an das System nicht benötigt werden, da die Annahmen, auf denen die Anforderungen basierten, falsch sein werden.

Ich habe mich aktiv für diese Architektur entschieden da ich erkannt habe, dass sich die Ressourcenanforderungen an die einzelnen Teile der Applikation mit zunehmender Nutzerzahlen unterschiedlich entwickeln werden. Zudem bot sich aufgrund der unterschiedlichen Funktionalitäten ein funktionaler Schritt an. Und genau hier bietet sich die Modulith-Architekt an.

Der entsprechenden ADR ist im Unterabschnitt 9.1.1 Monolith versus Micro-Services versus Modularer Monolith (Modulith) beschrieben.

5.4.1.1. Datenbank Server

Jedes Modul der Middleware verfügt über eine eigene Datenbank. Zudem handelt es sich bei den meisten dieser Datenbanken um Relationale PostgreSQL Datenbanken.

Diese PostgreSQL Datenbanken befinden sich aktuell alle auf demselben Datenbank-Server, um den Verwaltungsaufwand und die Kosten überschaubar zu halten.

Der entsprechende ADR ist im Unterabschnitt 9.1.8 - Ein Datenbankserver für die Relationalen Datenbanken der Middleware beschrieben.

5.4.1.2. API-Architektur

Die Endpunkte der Middleware-API können in zwei Kategorien von Endpunkten aufgeteilt werden. So folgen die meisten Endpunkte des Funktions-Modules «Command-Suite» dem „Processing resource“-Pattern aus dem Buch und der dazugehörigen Webseite «*Patterns for API Design*» (Zimmermann, Stocker, Lübke, Zdun, & Pautasso, Patterns for API Design, 2022). Dabei handelt es sich meist um Endpunkte, welche das Ziel verfolgen eine Aktion auszulösen. Im Anwendungsfall dieser Arbeit geht es um das Auslösen eines Events der zum Beispiel eine Lampe einschaltet.

Information Holder Resource-Pattern

Die meisten der anderen Endpunkte aus den Funktionsmodulen sind eher dem Pattern «*Information Holder Resource*» aus dem Buch und der dazugehörigen Webseite zuzuordnen (Zimmermann, Stocker, Lübke, Zdun, & Pautasso, Patterns for API Design, 2022). Bei denen geht es nicht darum eine Aktion auszulösen, sondern um den Austausch von inhaltlichen Informationen.

Atomic Parameter-Pattern

Bei allen Endpunkten, bei denen eine bestimmte Ressource abgefragt wird, wurde das «Atomic Parameter»-Pattern, auch aus dem zuvor genannten Buch und der dazugehörigen Webseite (Zimmermann, Stocker, Lübke, Zdun, & Pautasso, Patterns for API Design, 2022), angewandt. Hierbei wird jeweils die Objekt-Id als Parameter verwendet.

Zudem enthalten alle API-Antworten der Middleware, HATEOAS-links zu weitergehenden Informationen. Dies soll die Developer- Experience der Zielgruppe User-Interface Entwickler steigern.

5.4.1.2.1. GET-Endpoints

Da die Datenmodelle zum aktuellen Zeitpunkt noch überschaubar sind, wurde das «Embedded Entity-Pattern (Zimmermann, Stocker, Lübke, Zdun, & Pautasso, Patterns for API Design, 2022), welches alle Informationen der Entität enthält, dem «Linked Information Holder»-Pattern (Zimmermann, Stocker, Lübke, Zdun, & Pautasso, Patterns for API Design, 2022), welches anstelle der Information selbst einen Link zu den Informationen zur Verfügung stellt, vorgezogen. Beide

genannten Pattern stammen auch aus dem Buch und der dazugehörigen Webseite «*Patterns for API Design*».

Diese Entscheidung wurde aktiv getroffen, um den Aufwand bei der Entwicklung der Endpoints so gering als möglich zu halten. Denn somit konnte auf dezidierte DTO's verzichtet werden, was Performance- und Entwicklungs-Ressourcen spart.

Mehrseitige Abfragen
Bei allen Abfragen, bei welcher mehr als nur eine Ressource zurückgegeben wird, können folgende Darstellungs-Informationen zusätzlich mit der Anfrage mitgegeben werden:

- Attribut nach welchem die Ressourcen sortiert werden sollen
- Anzahl der Ressourcen-Instanzen die bei der Antwort mitgegeben werden sollen
- Welcher Abschnitt und somit welche Rangierungen der Ressourcen-Instanzen bei der Antwort mitgeliefert werden sollen

Dies entspricht dem Pagination-Pattern (Zimmermann, Stocker, Lübke, Zdun, & Pautasso, Patterns for API Design, 2022).

5.4.1.2.2. POST-Endpoints

Beim Erstellen von neuen Ressourcen wurden teilweise Datentransfer-Objekte eingesetzt, um die vom Endbenutzer verlangten Informationen auf ein Minimum zu reduzieren.

Die Antworten der POST-Endpoints enthalten jeweils den Identifier der neu erstellten Ressource. Falls das Interesse besteht die gesamte Ressource einzusehen, so kann dies über den mitgelieferten HEATOS-Link und dem Identifier der Ressource gemacht werden. Die Alternative dazu wäre gewesen das gesamte neu erstellte Objekt zurückzuliefern. Darauf wurde unter der Berücksichtigung der Übertragungsressourcen verzichtet.

5.4.1.3. API-Endpunkte

Folgende Endpunkte stehen dem User-Interface Entwickler aktuell zur Verfügung:

- *GET /api/v1/user-management/my-user*
- *PUT /api/v1/user-management/my-user*
- *PATCH /api/v1/user-management/my-user*
- *GET /api/v1/automation/triggers/{triggerId}*
- *PUT /api/v1/automation/triggers/{triggerId}*
- *DELETE /api/v1/automation/triggers/{triggerId}*
- *POST /api/v1/monitoring/triggers/user_trigger*
- *POST /api/v1/monitoring/trigger/device_trigger*
- *POST /api/v1/monitoring/trigger/connector_trigger*
- *GET /api/v1/automation/triggers*
- *POST /api/v1/automation/triggers*
- *GET /api/v1/monitoring/triggers*
- *DELETE /api/v1/monitoring/trigger*
- *GET /api/v1/automation/policy/{policyId}*
- *PUT /api/v1/automation/policy/{policyId}*
- *DELETE /api/v1/automation/policy/{policyId}*
- *GET /api/v1/automation/policy*
- *POST /api/v1/automation/policy*
- *GET /api/v1/automation/effect/{effectId}*
- *PUT /api/v1/automation/effect/{effectId}*
- *DELETE /api/v1/automation/effect/{effectId}*
- *GET /api/v1/automation/effect*
- *POST /api/v1/automation/effect*
- *POST /api/v1/user-management/connector/reconnect*
- *POST /api/v1/user-management/connector/initialize*
- *POST /api/v1/control/power-device/execute-command*
- *GET /api/v1/control/power-devices*
- *POST /api/v1/control/music-device/execute-command*
- *GET /api/v1/control/music-devices*
- *GET /api/v1/control/devices*
- *POST /api/v1/control/devices*
- *POST /api/v1/control/cover-device/execute-command*
- *GET /api/v1/control/cover-devices*
- *POST /api/v1/authentication/register*
- *POST /api/v1/authentication/refresh-token*
- *POST /api/v1/authentication/authenticate*
- *GET /api/v1/user-management/system/{deviceId}*
- *PATCH /api/v1/user-management/system/{deviceId}*
- *GET /api/v1/user-management/system/{connectorId}*
- *PATCH /api/v1/user-management/system/{connectorId}*
- *GET /api/v1/user-management/system/devices*
- *GET /api/v1/user-management/system/connectors*
- *GET /api/v1/monitoring/records*
- *GET /api/v1/monitoring/records/filtered*
- *GET /api/v1/monitoring/live/records/{subscriptionId}*

5.4.1.4. Data Distribution

Die genaue Repräsentation der Grund-Datentypen (User, Gerät und Konnektor) unterscheidet sich zwar zwischen den einzelnen Funktionsmodulen, doch sind diese Grund-Datentypen in den meisten Funktionsmodulen vorhanden. Daher werden bei einer Anpassung der Grund-Datentypen diese mittels Application-Event-Publisher an die Module übermittelt. Dies geschieht mittels «DistributedDataMessage».

Aufbau der «DistributedDataMessage»-Nachricht

Folgende Informationen werden in der Nachricht übertragen:

- Zeitstempel zum Zeitpunkt des Erstellens der Update-Nachricht
- Objekt-Identifikator des Update-Objektes
- neuer Datenstand in der Repräsentation des Grund-Datentyps

Basierend auf dem Objekt-Identifikator wird das jeweilige Objekt in der Modul-Datenbank gesucht. Falls dieses nicht gefunden wird, so wird es nach dem Umwandeln in die modul-spezifische Repräsentation umgewandelt und im Anschluss gespeichert. Falls es gefunden wird, so werden die Attribute, welche sowohl auf der modul-spezifischen Repräsentation als auch auf dem Grund-Datentyp enthalten sind, verglichen und bei Unterschieden dem eigenen Datenbestand angepasst.

5.4.2. ActiveMQ Artemis

Als Messaging-Plattform wurde ActiveMQ Artemis eingesetzt. Dabei handelt es sich um ein Projekt der Apache Software Foundation.

Protokoll

Da ActiveMQ Artemis mehrere Übertragungsprotokolle unterstützt, fiel die Wahl auf das Core-Protokoll, welches die Daten als JSON serialisiert überträgt. Die Hintergründe der Entscheidung sind im ADR des Unterabschnittes 9.1.19 - Wahl des ActiveMQ Artemis Protokolls beschrieben.

Deployment

Diese Instanz wird aus Kostengründen in einem Docker-Container auf Azure Container Instance betrieben. *Weitere Einzelheiten diesbezüglich sind im Unterabschnitt 7.1.4.3 Messaging Plattform ausgeführt.*

5.4.3. Command Suite

Die Aufgabe der Command Suite besteht darin, Anweisungen vom User-Interface an den entsprechenden Konnektor zu versenden.

5.4.3.1. Modul-Einblick in das Modul Command-Suite

Modulübersicht

Nachstehend wird das Container Diagramm des Modules Command Suite dargestellt:

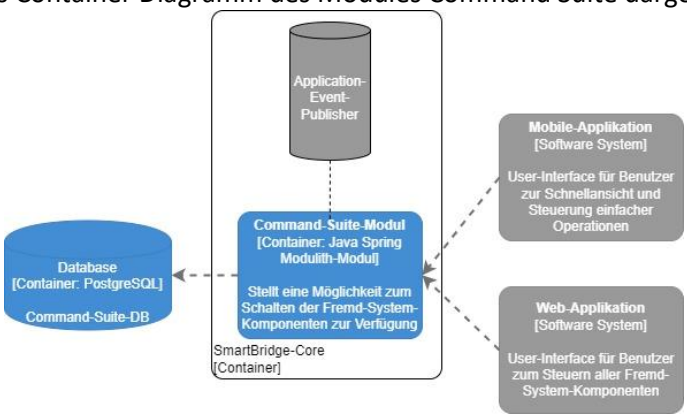


Abbildung 43: Container Diagramm des Modules Command-Suite

Detaileinsicht

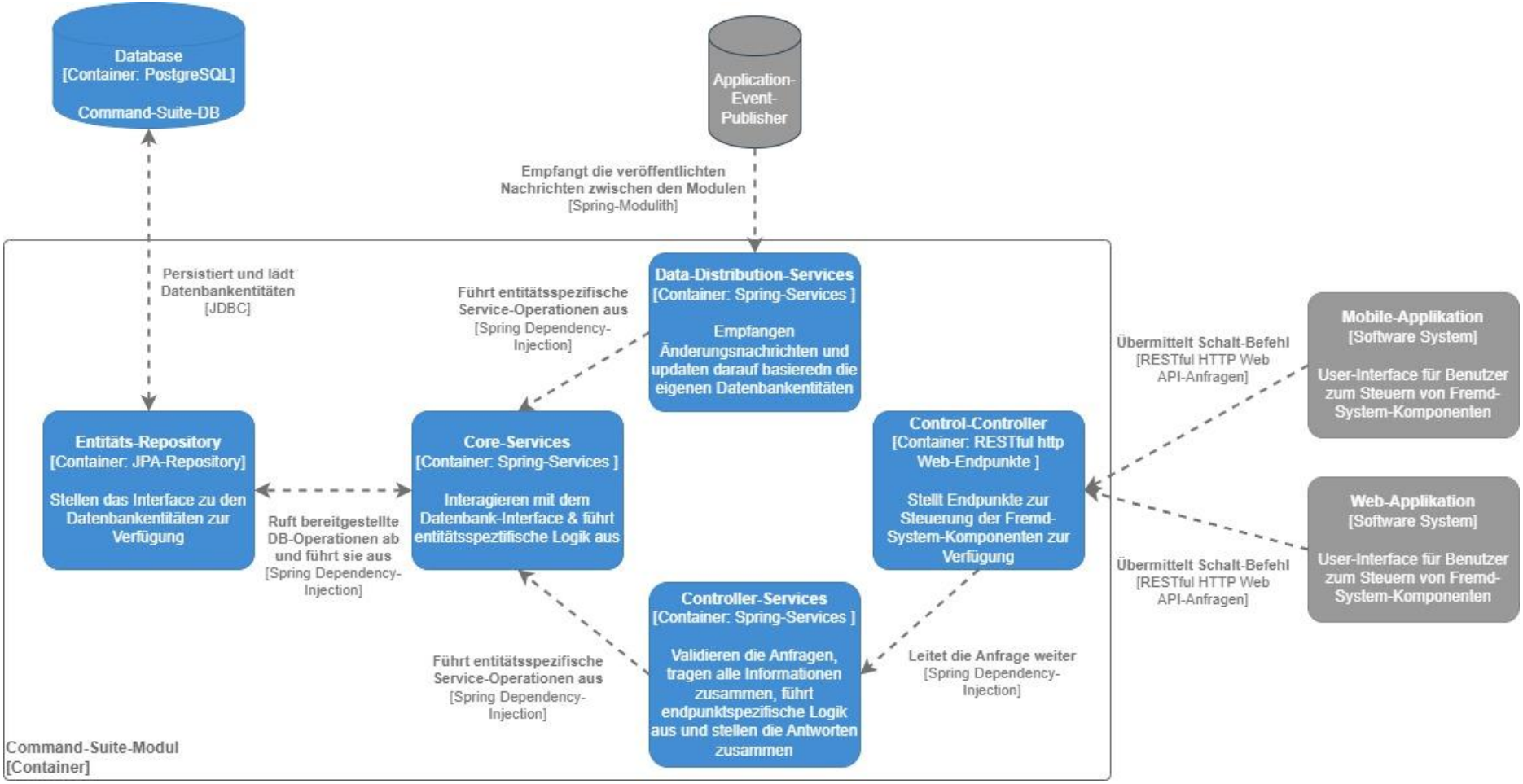


Abbildung 44: Komponenten Diagramm des Modules Command-Suite

5.4.3.2. Daten-Update der Grund-Datentypen im Modul Command-Suite

In den folgenden Abschnitten wird das Mapping der Attribute zwischen den Grund-Datentypen und der Command-Suite-spezifischen Repräsentation ausgeführt. *Weitere Informationen bezüglich dem Data-Distribution-Mechanismus sind im Unterabschnitt 5.4.1.4 - D*ata Distribution *genauer erläutert.*

5.4.3.2.1. User-Mapping im Modul Command-Suite

Die folgende Tabelle stellt diejenigen Felder dar, die zwischen der Grund-Datentyp-Repräsentation des Users und der Datentyp-Repräsentation innerhalb der Command-Suite synchronisiert wird.

Geteilte Repräsentation	**Synchronisation**	**Moduleigene Repräsentation**
id	Nein	id
userId	ja	userId
email	Nein	N/A
password	Nein	N/A
firstName	ja	firstName
lastName	ja	lastName
dateOfBirth	Nein	N/A
role	Nein	N/A
tokens	Nein	N/A
ownedConnectors	Nein	N/A
viewedConnectors	Nein	N/A
N/A	Nein	devices

Tabelle 4: Übersicht des User-Mappings im Modul Command-Suite

5.4.3.2.2. Device-Mapping im Modul Command-Suite

Die folgende Tabelle stellt diejenigen Felder dar, die zwischen der Grund-Datentyp-Repräsentation des Devices und der Datentyp-Repräsentation innerhalb der Command-Suite synchronisiert wird.

Geteilte Repräsentation eines Gerätes	**Synchronisation**	**Moduleigene Repräsentation eines Gerätes**
id	Nein	id
deviceId	ja	deviceId
name	Ja	deviceName
description	Nein	N/A
isSensor	Nein	N/A
connector	Nein	N/A
N/A	Nein	deviceType
N/A	Nein	connectorId
N/A	Nein	user

Tabelle 5: Übersicht des Device-Mappings im Modul Command-Suite

5.4.3.2.3. Connector-Mapping im Modul Command-Suite

Das Modul Command Suite verfügt über keine Repräsentation des Konnektors. Einzig die Konnektor-ID wird auf dem Device gespeichert, da es für die korrekte Adressierung einer Command-Message benötigt wird. Ansonsten ist das Konzept des Konnektors irrelevant für die Command Suite.

5.4.4. Monitoring Suite

Die Monitoring Suite soll dem Endbenutzer einen Live-Einblick in seine Systeme geben. So kann er die letzten Nachrichten, die auf den Fremd-Systemen gesendet wurden, einsehen.

5.4.4.1. Monitor-Message-Queue

Damit die neu hinzukommenden Meldungen der Fremd-Systeme zeitaktuell zur Hand sind, werden diese in eine Ring-Buffer-ähnliche Queue gespeichert. Um ein unnötiges Kopieren zu vermeiden, wurde der Ring-Buffer mittels Linked-Blocking-Queue umgesetzt, welche jedoch mit einer maximalen Kapazität versehen wurde.

5.4.4.2. Modul-Einblick in das Modul Monitoring-Suite

Nachstehend wird das Cointainer Diagramm des Modules Monitoring-Suite wir folgt dargestellt:

Modulübersicht

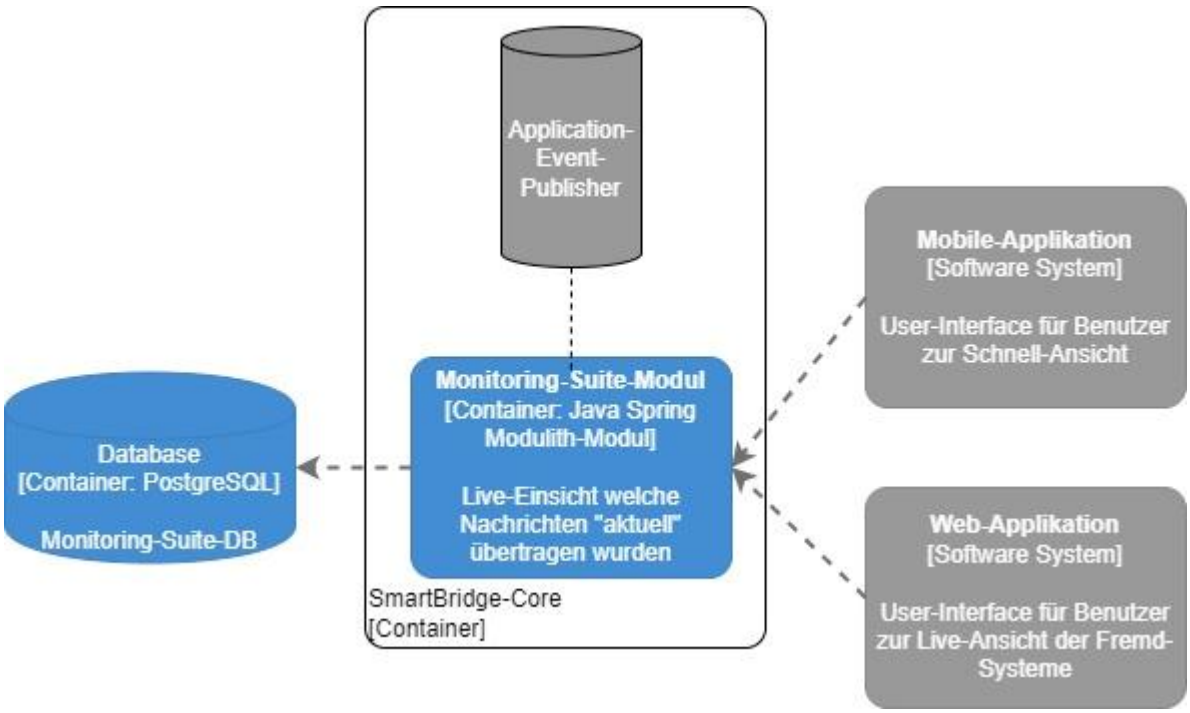


Abbildung 45: Container Diagramm des Modules Monitoring-Suite

Detaileinsicht

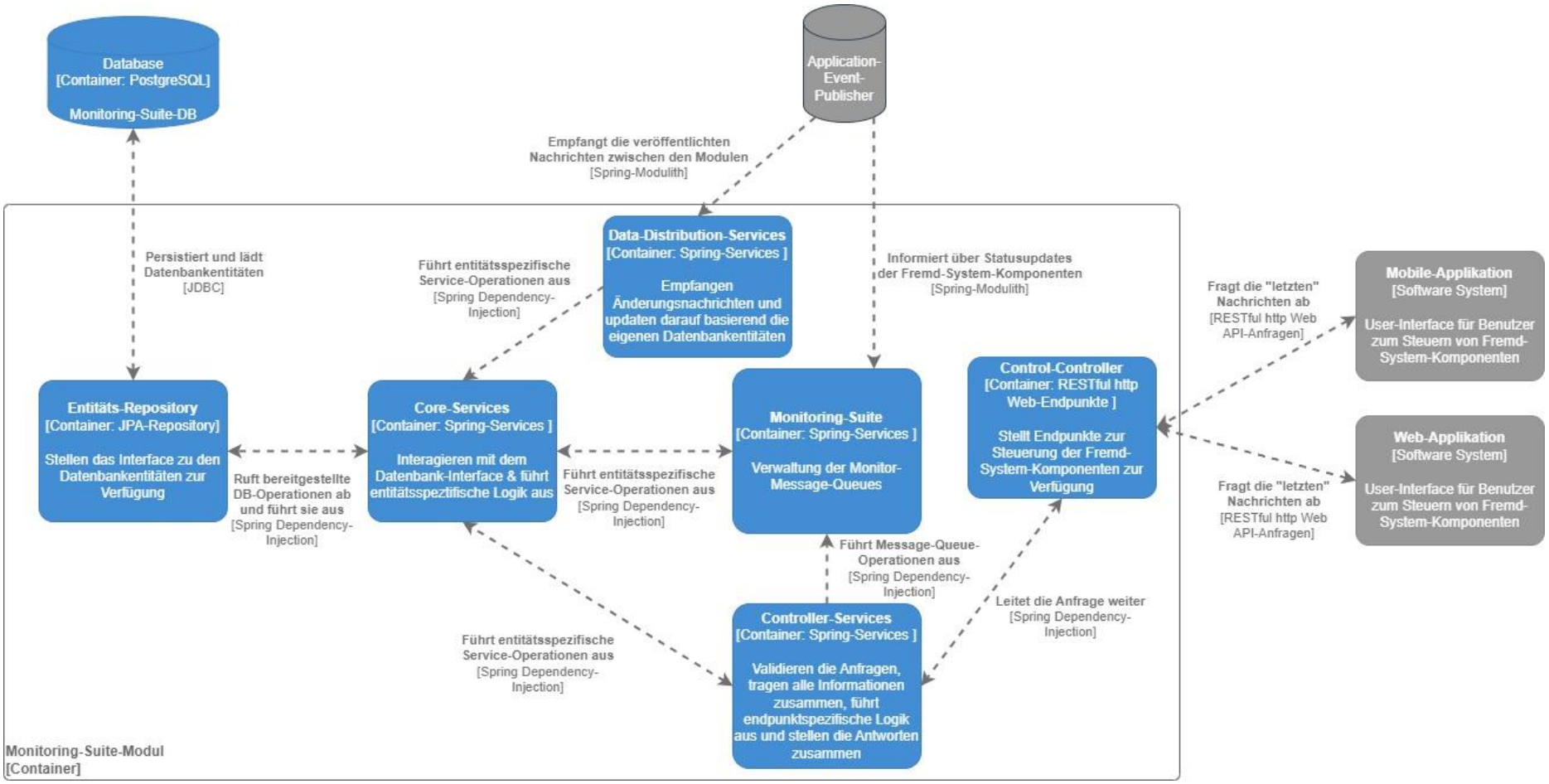


Abbildung 46: Komponenten Diagramm des Modules Monitoring-Suite

5.4.4.3. Daten-Update der Grund-Datentypen im Modul Monitoring-Suite

In den folgenden Abschnitten wird das Mapping der Attribute zwischen den Grund-Datentypen und der Monitoring-Suite-spezifischen Repräsentation ausgeführt. *Weitere Informationen bezüglich dem Data-Distribution-Mechanismus sind im Unterabschnitt 5.4.1.4 - D*ata Distribution *genauer erläutert.*

5.4.4.3.1. User-Mapping im Modul Monitoring-Suite

Die folgende Tabelle stellt diejenigen Felder dar, die zwischen der Grund-Datentyp-Repräsentation des Users und der Datentyp-Repräsentation innerhalb der Monitoring-Suite synchronisiert wird.

Geteilte Repräsentation	**Synchronisation**	**Moduleigene Repräsentation**
id	Nein	id
userId	Ja	userId
email	Nein	N/A
password	Nein	N/A
firstName	Nein	N/A
lastName	Nein	N/A
dateOfBirth	Nein	N/A
role	Nein	N/A
tokens	Nein	N/A
ownedConnectors	Nein	N/A
viewedConnectors	Nein	N/A

Tabelle 6: Übersicht des User-Mappings im Modul Monitoring-Suite

5.4.4.3.2. Device-Mapping im Modul Monitoring-Suite

Die folgende Tabelle stellt diejenigen Felder dar, die zwischen der Grund-Datentyp-Repräsentation des Devices und der Datentyp-Repräsentation innerhalb der Monitoring-Suite synchronisiert wird.

Geteilte Repräsentation	**Synchronisation**	**Moduleigene Repräsentation**
id	Nein	id
deviceId	Ja	deviceId
name	Ja	deviceName
description		N/A
isSensor	Ja	isSensor
connector	Ja	connector
N/A	Nein	N/A
N/A	Nein	N/A

Tabelle 7: Übersicht des Device-Mappings im Modul Device-Suite

5.4.4.3.3. Connector-Mapping im Modul Monitoring-Suite

Die folgende Tabelle stellt diejenigen Felder dar, die zwischen der Grund-Datentyp-Repräsentation des Connectors und der Datentyp-Repräsentation innerhalb der Monitoring-Suite synchronisiert wird.

Geteilte Repräsentation	**Synchronisation**	**Moduleigene Repräsentation**
id	nein	id
connectorId	ja	connectorId
connectorName	Nein	N/A
connectorType	Nein	N/A
connectorDescription	Nein	N/A
latitude	Nein	N/A
longitude	Nein	N/A
mainUser	ja	user

Tabelle 8: Übersicht des Connector-Mappings im Modul Devi

5.4.5. Policy Hub

Die Aufgabe des Policy Hub ist es, automatisiert Events auszulösen. Hierbei gibt es bisher folgende Auslöser-Typen:

- **Datum- und zeitbasierte Auslöser**
  Datum- und zeitbasierte Auslöser stellen Events dar die ausgelöst werden, da aktuell die vom Endbenutzer hinterlegte Zeit bei seinem Fremd-System erreicht wurde. Hierbei wird auch auf die Zeitzone des Fremd-Systems geachtet.
  Ein möglicher Anwendungsfall hierfür ist das Einrichten einer zeitschaltur-basierten Jalousien-Steuerung, ohne dass eine zusätzliche Hardware-Komponente angeschafft werden muss.

- **Event-basierte Auslöser**
  Bei den event-basierten Auslöser liegt ein anderer Event zugrunde. Der auslösende Event kann jedoch auch von einem anderen Fremd-System kommen.
  So wäre ein möglicher Anwendungsfall, dass sobald das Licht im gesamten Erdgeschoss über eine Zentralfunktion ausgeschaltet werden möchte auch die Lautsprecher-Boxen in der Küche die Musik pausieren soll.

5.4.5.1. Modul-Einblick in das Modul Policy-Hub

Nachstehend wird das Container Diagramm des Modules Policy-Hub als Modulübersicht dargestellt:

Modulübersicht

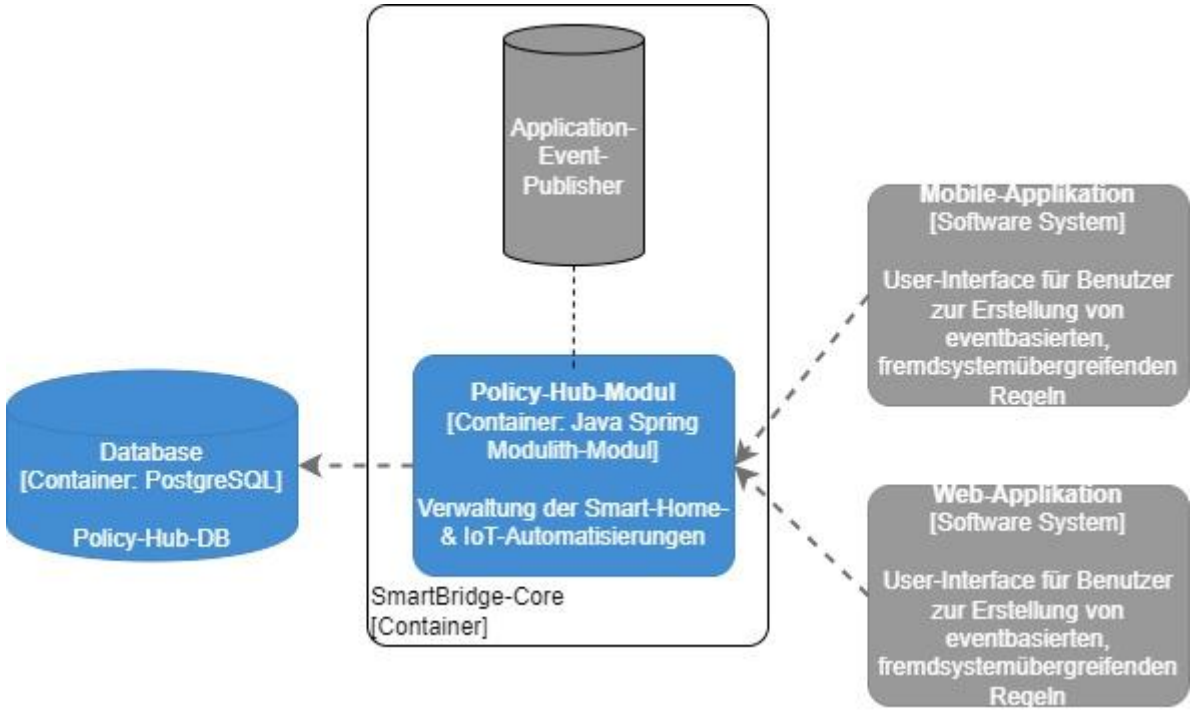


Abbildung 47: Container Diagramm des Modules Policy-Hub

Detaileinsicht

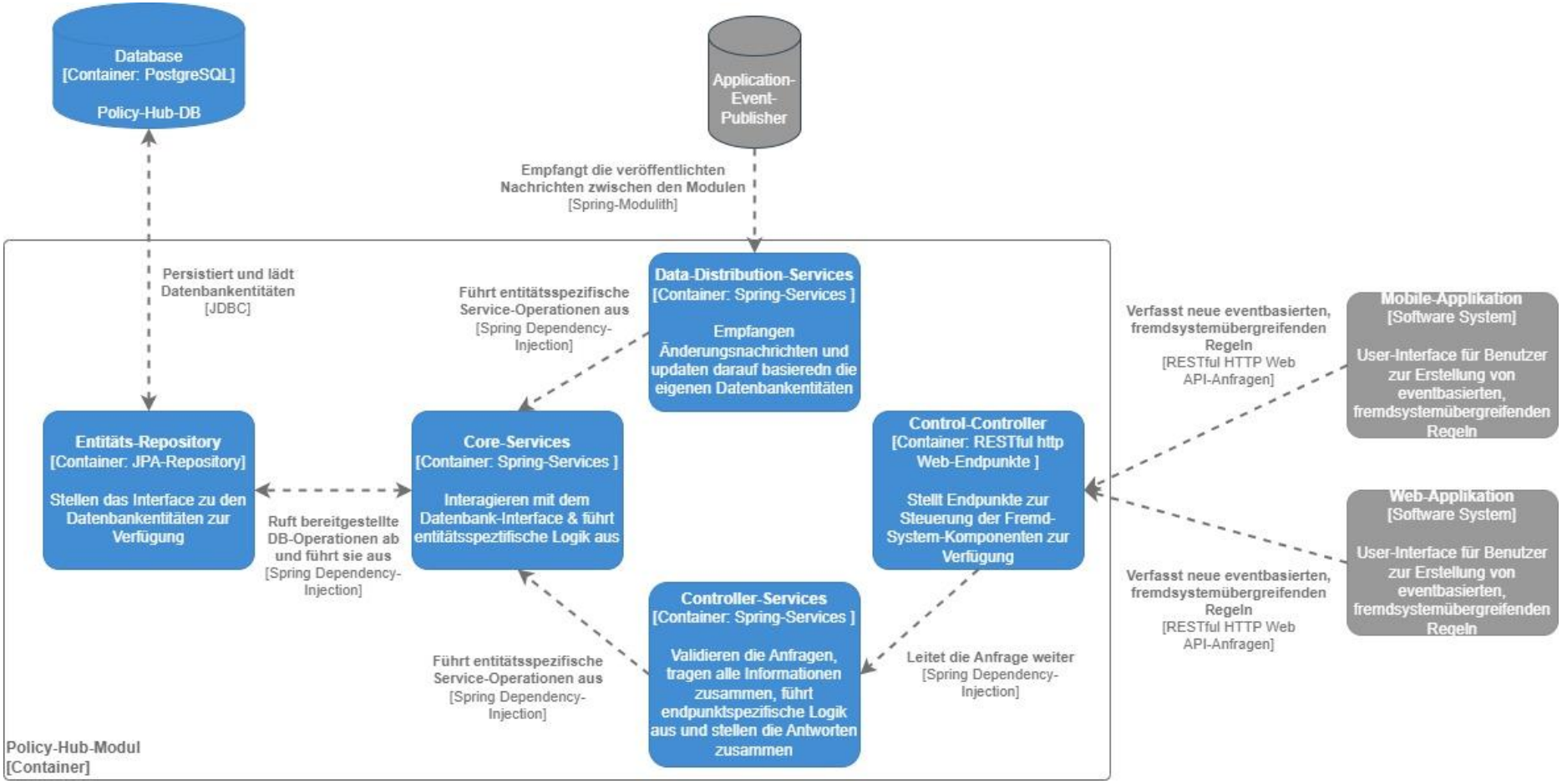


Abbildung 48: Komponenten Diagramm des Modules Policy-Hub

5.4.5.2. Daten-Update der Grund-Datentypen im Modul Policy-Hub

In den folgenden Unterabschnitten wird das Mapping der Attribute zwischen den Grund-Datentypen und der Policy-Hub-spezifischen Repräsentation ausgeführt. *Weitere Informationen bezüglich dem Data-Distribution-Mechanismus sind im Unterabschnitt 5.4.1.4 - D*ata Distribution *genauer erläutert.*

5.4.5.2.1. User-Mapping im Modul Policy-Hub

Die folgende Tabelle stellt diejenigen Felder dar, die zwischen der Grund-Datentyp-Repräsentation des Users und der Datentyp-Repräsentation innerhalb der Policy-Hub synchronisiert wird.

Geteilte Repräsentation	**Synchronisation**	**Moduleigene Repräsentation**
id	Nein	id
userId	Ja	userId
email	Ja	email
password	Nein	N/A
firstName	Nein	N/A
lastName	Nein	N/A
dateOfBirth	Nein	N/A
role	Nein	N/A
tokens	Nein	N/A
ownedConnectors	Nein	N/A
viewedConnectors	Nein	N/A
N/A	Nein	Policies
N/A	Nein	connectors

Tabelle 9: Übersicht des User-Mappings im Modul Policy-Hub

5.4.5.2.2. Device-Mapping im Modul Policy-Hub

Die folgende Tabelle stellt diejenigen Felder dar, die zwischen der Grund-Datentyp-Repräsentation des Devices und der Datentyp-Repräsentation innerhalb der Policy-Hub synchronisiert wird.

Geteilte Repräsentation	**Synchronisation**	**Moduleigene Repräsentation**
id	Nein	id
deviceId	Ja	deviceId
name	Nein	N/A
description	Nein	N/A
isSensor	Nein	N/A
connector	(ja)	connectorId

Tabelle 10: Übersicht des Device-Mappings im Modul Policy-Hub

5.4.5.2.3. Connector-Mapping im Modul Policy-Hub

Die folgende Tabelle stellt diejenigen Felder dar, die zwischen der Grund-Datentyp-Repräsentation des Connector und der Datentyp-Repräsentation innerhalb der Policy-Hub synchronisiert wird.

Geteilte Repräsentation	**Synchronisation**	**Moduleigene Repräsentation**
id	Nein	N/A
connectorId	Nein	N/A
connectorName	Nein	N/A
connectorType	Nein	N/A
connectorDescription	Nein	N/A
latitude	Nein	N/A
longitude	Nein	N/A
mainUser	Nein	N/A

Tabelle 11: Übersicht des Connector-Mappings im Modul Policy-Hub

5.4.6. Record Keeper

Der Record-Keeper wurde mit der Aufgabe betraut, alle Nachrichten von einer bestimmten Quelle zu speichern. Zudem können die Quellen auf den folgenden verschiedenen Granularitäts-Ausprägungen gewählt werden:

- **Gerät**
  Bei der Auswahl eines spezifischen Geräts werden alle Nachrichten, die von diesem Gerät stammen, gespeichert.

- **Konnektor**
  Wenn sich der Endbenutzer dazu entscheiden sollte alle Nachrichten, die von einem bestimmten Konnektor stammen zu speichern, so werden alle Nachrichten aller Geräte, die sich hinter diesem Konnektor befinden, persistiert.

- **Account**
  Wenn sich der Endbenutzer entscheidet alle Nachrichten zu speichern, werden alle Nachrichten aller Geräte hinter allen Konnektoren gespeichert.

Bei der Architektur und der Entwicklung des Record-Keepers wurde darauf geachtet, dass es ohne grossen Aufwand möglich wäre alle Nachrichten, die zu einem bestimmten Gerät oder Konnektor respektive innerhalb eines Accounts gesendet werden, zu speichern. Dies wäre jedoch eine Erweiterung und wurde in der Arbeit nicht umgesetzt.

1.1.1.1. Modul-Einblick in das Modul Recod-Keeper

Nachstehend wird das Container Diagramm des Modules Record Keeper dargestellt:

Modulübersicht

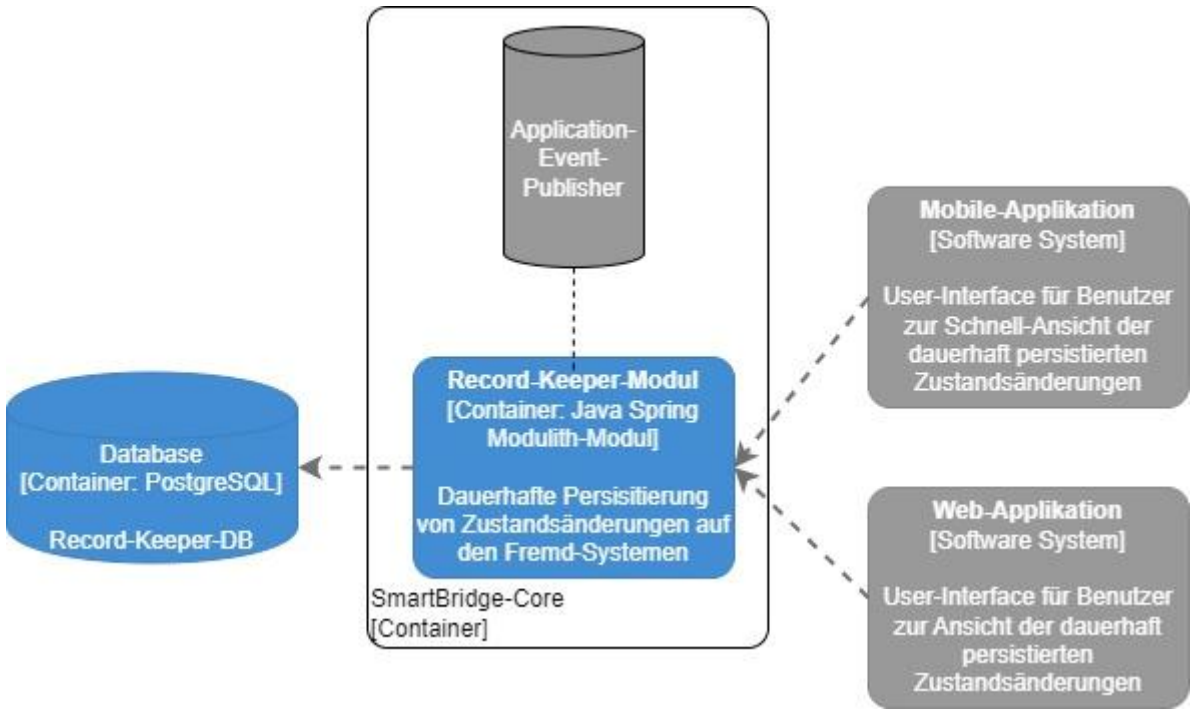


Abbildung 49: Container Diagramm des Modules Record-Keeper

Detaileinsicht

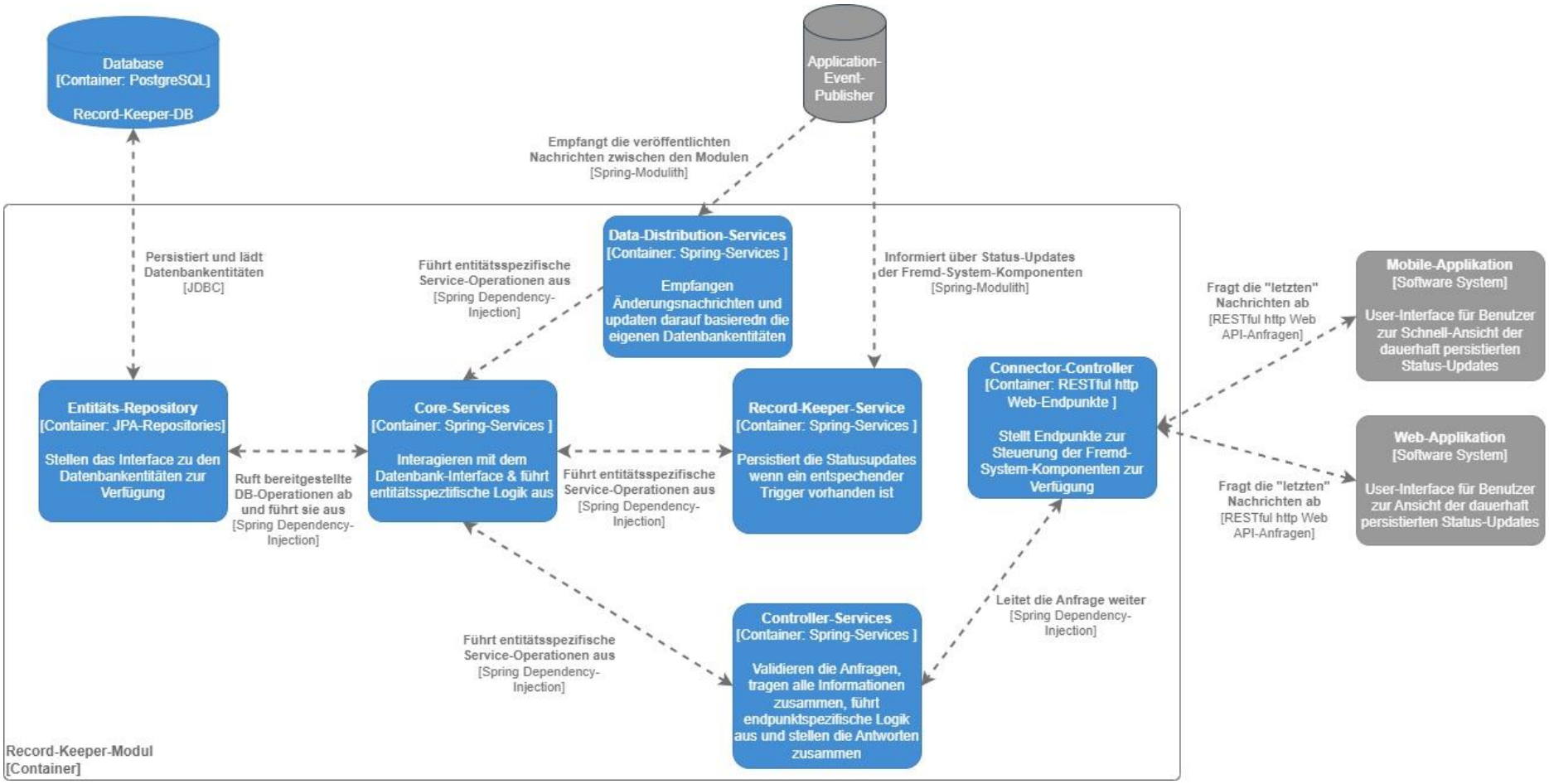


Abbildung 50: Komponenten Diagramm des Modules Record-Keeper

5.4.6.1. Daten-Update der Grund-Datentypen im Modul Record-Keeper

In den folgenden Unterabschnitten wird das Mapping der Attribute zwischen den Grund-Datentypen und der Record-Keeper-spezifischen Repräsentation ausgeführt. *Weitere Informationen bezüglich dem Data-Distribution-Mechanismus sind im Unterabschnitt 5.4.1.4 - D*ata Distribution *genauer erläutert.*

5.4.6.1.1. User-Mapping im Modul Record-Keeper

Die folgende Tabelle stellt diejenigen Felder dar, die zwischen der Grund-Datentyp-Repräsentation des Users und der Datentyp-Repräsentation innerhalb der Record-Keeper synchronisiert wird.

Geteilte Repräsentation	**Synchronisation**	**Moduleigene Repräsentation**
id	Nein	id
userId	Ja	userId
email	Ja	email
password	Nein	N/A
firstName	Nein	N/A
lastName	Nein	N/A
dateOfBirth	Nein	N/A
role	Nein	N/A
tokens	Nein	N/A
ownedConnectors	Ja	connectors
viewedConnectors	Nein	N/A

Tabelle 12: Übersicht des User-Mappings im Modul Record-Keeper

5.4.6.1.2. Device-Mapping im Modul Record-Keeper

Die folgende Tabelle stellt diejenigen Felder dar, die zwischen der Grund-Datentyp-Repräsentation des Devices und der Datentyp-Repräsentation innerhalb der Record-Keeper synchronisiert wird.

Geteilte Repräsentation	**Synchronisation**	**Moduleigene Repräsentation**
id	Nein	N/A
deviceId	Nein	N/A
name	Nein	N/A
description	Nein	N/A
isSensor	Nein	N/A
connector	Nein	N/A
id	Nein	N/A
deviceId	Nein	N/A

Tabelle 13: Übersicht des Device-Mappings im Modul Record-Keeper

5.4.6.1.3. Connector-Mapping im Modul Record-Keeper

Die folgende Tabelle stellt diejenigen Felder dar, die zwischen der Grund-Datentyp-Repräsentation des Connectors und der Datentyp-Repräsentation innerhalb der Record-Keeper synchronisiert wird.

Geteilte Repräsentation	**Synchronisation**	**Moduleigene Repräsentation**
id	Nein	id
connectorId	Ja	connectorId
connectorName	Nein	N/A
connectorType	Nein	N/A
connectorDescription	Nein	N/A
latitude	Nein	N/A
longitude	Nein	N/A
mainUser	(Ja)	userId

Tabelle 14: Übersicht des Connector-Mappings im Modul Record-Keeper

5.4.7. Common

Die primäre Funktion dieses Moduls liegt in der Bereitstellung von allgemeinen Diensten, Datenstrukturen und Konfigurationen, die von mehreren anderen Modulen benötigt werden. Es dient dazu, Code-Duplizierungen zu vermeiden und eine konsistente Nutzung gemeinsamer Elemente über verschiedene Module hinweg zu ermöglichen.

5.4.7.1. Modul-Einblick

Nachstehend wird das Container Diagramm des Modules Common dargestellt:

Modulübersicht

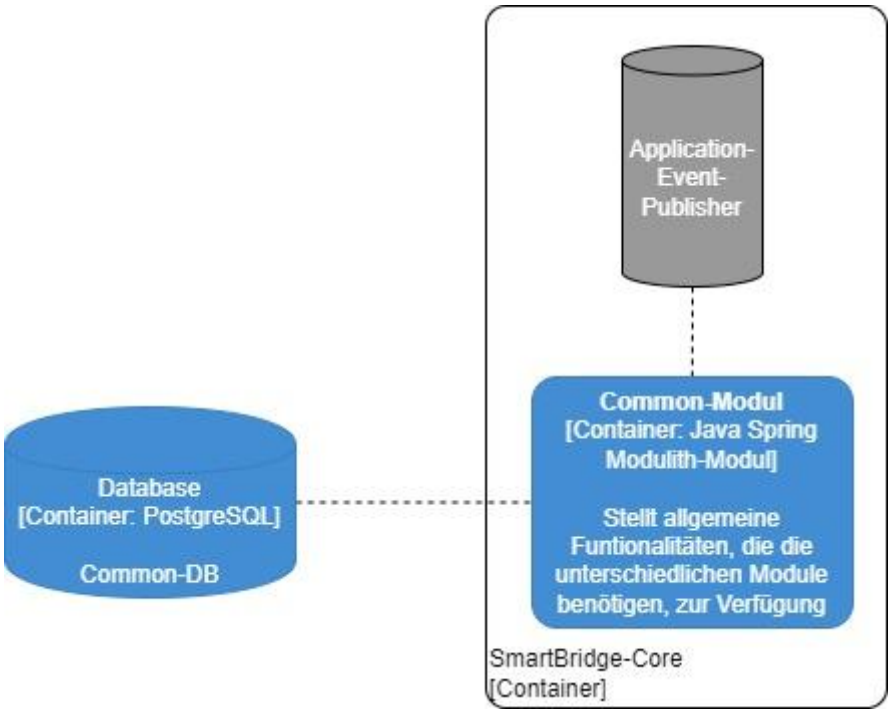


Abbildung 51: Container Diagramm des Modules Common

Detaileinsicht

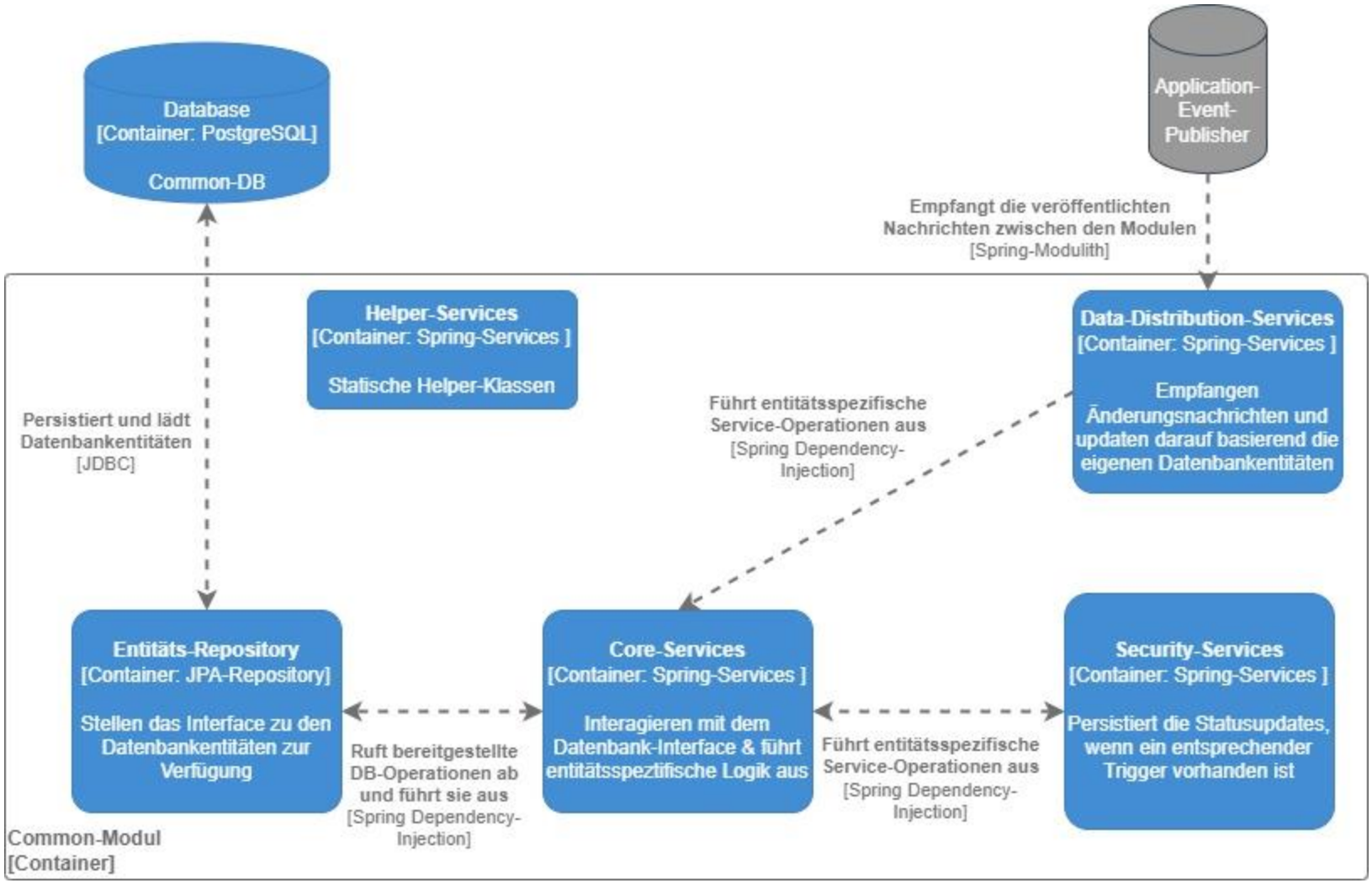


Abbildung 52: Komponenten Diagramm des Modules Common

5.4.7.2. Daten-Update

Im Gegensatz zu den anderen Modulen speichert das Common Modul die meisten Daten in der Grund-Repräsentation.

5.4.7.2.1. User-Mapping

Die User-Repräsentation innerhalb des Common-Moduls baut auf der Grund-Datenrepräsentation des User-Objektes auf und wird durch die User-Details des Spring-Security Frameworks erweitert. Dies wird für die spring-interne Authentication und Autorisationslösung benötigt. Diese ist auch für das Ausstellen und die Validierung der Zugriffs- und Refresh-Token zuständig.

5.4.7.2.2. Device-Mapping

Das Common-Modul persistiert keine Device-Objekte in der modul-spezifischen Datenbank.

5.4.7.2.3. Connector-Mapping

Um auch nach einem Neustart der Applikation die Verbindungen zu den bestehenden Konnektoren wieder aufzunehmen, werden die Konnektor-Informationen in der modul-spezifischen Datenbank des Common-Moduls gespeichert.

Dies geschieht in der Grund-Datenrepräsentation und benötigt daher kein Mapping der Datentypen.

5.4.7.2.4. Token-Mapping

Die Validierung der Access-Token wird zentral für alle API-Endpunkte registriert. Daher müssen diese neben den Usern und den Konnektoren auch in der Common-Datenbank persistiert werden. Dies geschieht auch in der Grund-Datenrepräsentation.

5.4.8. Orchestration

Die Hauptaufgabe des Modules Orchestration besteht darin, das Handling der Nachrichten vorzunehmen. So wandelt es die internen Events um und sendet sie an den entsprechenden Konnektor.

5.4.8.1. Modul-Einsicht

Das Orchestrations-Modul hat, nach dem «Single-Responsibility»-Principle nur eine Aufgabe, nämlich das Verpacken und Entpacken von Nachrichten, die von der Middleware an die Konnektoren oder in die entgegengesetzte Richtung versendet werden.

Sobald die Nachrichten mittels ActiveMQ Artemis übertragen werden, so geschieht dies in Form einer «SmartBridgeJmsMessage». *Der genaue Aufbau und die enthaltenen Felder dieser Nachrichten sind im Unterabschnitt 4.2.4 - Messaging Message beschrieben.*

Das Verpacken der eigentlichen Nachrichten in die «SmartBridgeJmsMessage» folgt dem «Envelope Wrapper»-Pattern (Hohpe & Woolf , Envelope Wrapper, 2003) aus dem Buch und der gleichnamigen Webseite «Enterprise Integration Patterns».

Modulübersicht

Nachstehend wird das Container Diagramm des Modules Orchestration dargestellt:

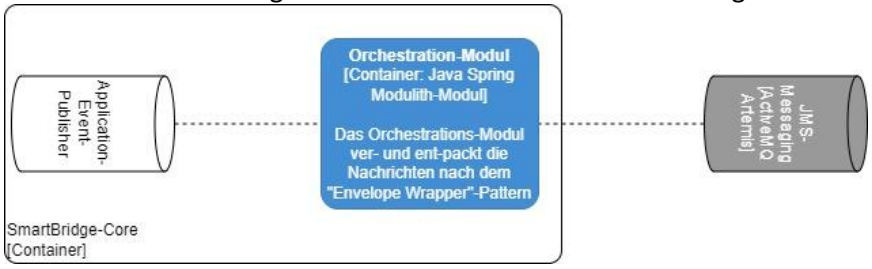


Abbildung 53: Container Diagramm des Modules Orchestration

5.5. Demo User-Interface Applikation

Im Rahmen dieser Arbeit wurde auch ein User-Interface für Demonstrationszwecke erstellt. Hierzu wurde ein Web-basiertes User-Interface auf der Grundlage des Java Frameworks Vaadin erstellt.

5.5.1. Vaadin

Vaadin ist ein java-basiertes Frontend-Framework, das für die Entwicklung von Web-Applikationen genutzt werden kann. Es wurde unter der Apache 2.0 Lizenz veröffentlicht und bietet eine serverseitige Architektur an. Der Vorteil von Vaadin ist, dass aus einer bereits vorgefertigten Palette von Komponenten und Views eine Frontend-Basis erstellt und dann individualisiert werden kann. Zudem bietet es die Option, das Aussehen und Verhalten sowohl in Java als auch in JavaScript zu beeinflussen.

Im Rahmen dieser Arbeit wird die Demo User-Interface Applikation als eine eigenständig laufende Software-Instanz betrieben. Dies stellt auch denselben Betriebsmodus für die zukünftigen User-Interface Implementationen dar.

5.5.2. Modul-Einsicht

Der Aufbau des Demo-Interfaces ist bewusst sehr simple gehalten. Die vorhandenen Kommunikations-Services, die Übermittlung und die Anfrage der Informationen und die Views stellen die eingeholten Informationen dar.

Nachstehend wird das Komponenten Diagramm des Demo-User-Interfaces dargestellt:

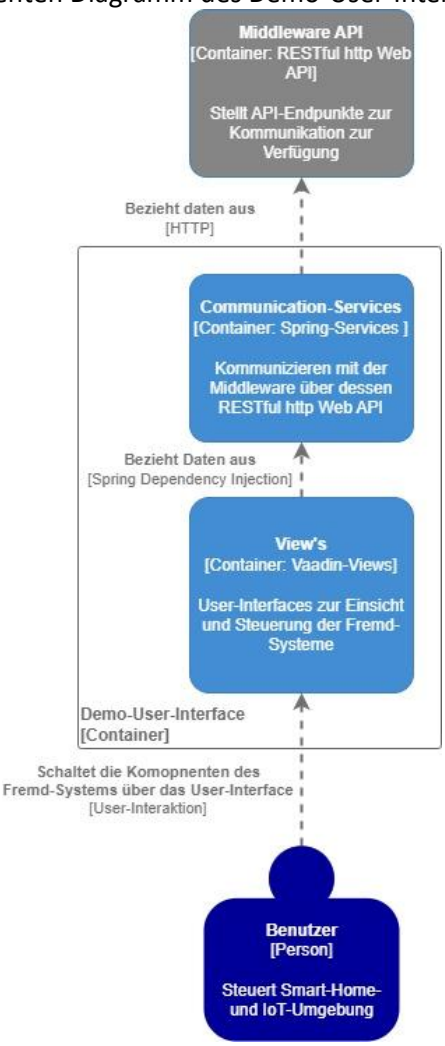


Abbildung 54: Komponenten Diagramm des Demo-User-Interfaces

6. Laufzeitsicht

In diesem Kapitel Laufzeit werden die verschiedenen Bestandteile des Gesamt-Systems aus der Laufzeit-Sicht erläutert. Dies bedeutet, dass der Fluss eines Events von Anfang bis Ende betrachtet wird.

Diejenigen Events, welche den normalen Betrieb darstellen, werden aus Sicht der Nachrichten betrachtet, da dies einen detaillierteren Einblick in den Event selbst bietet. Hingegen bei jenen Events, welche nur weniger oft durchgeführt werden, wird auf die Prozesssicht eingegangen. Dies aus dem Grund, dass bei den Prozessen mehr Wert auf den Ablauf als auf den Event-Inhalt gelegt wird.

6.1. Messages

Die in diesem Abschnitt beschriebenen Messages beziehen sich auf die Kernevents des Eco-Systems und behandeln die Übertragung der Status-Updates auf den Fremd-Systemen.

6.1.1. Übersicht der Nachrichten

In diesem Unterabschnitt werden die wichtigsten Messages-Typen nochmals erläutert. Mehr Informationen zu den jeweiligen Messages sind im Abschnitt 4.2 - Messages beschrieben.

6.1.1.1. Systemspezifische Nachrichten

Die System Messages sind jene Nachrichten, welche das Fremd-System dem Konnektor die Statusänderungen mitteilt. *Weitere Informationen in Bezug auf diese Nachrichten sind dem Unterabschnitt 4.2.2 - Systemspezifische Nachrichten zu entnehmen.*

6.1.1.2. System-Messages

Die Nachrichten, welche innerhalb der Konnektoren und auch der Middleware veröffentlicht werden, sind die System-Messages. *Weitere Informationen bezüglich dieser Nachrichten stehen im Unterabschnitt 4.2.3.2 - Third-Party-System-Message zur Verfügung.*

6.1.1.3. Messaging-Message

Die zuvor erwähnten System-Messages werden jeweils in eine «SmartBridgeJMSMessage» verpackt und versendet. *Informationen zum Aufbau dieser Nachrichten sind im Unterabschnitt 4.2.4 - Messaging Message aufgeführt.*

Event-Message

Wenn eine Message vom Konnektor versendet wird, so enthält die Nachricht das Topic «Event-Message». *Weiterführende Informationen bezüglich dieses Nachrichten-Typs sind im Unterabschnitt 4.2.1.5 - Event-Message ausgeführt.*

Controll-Command

Hingegen enthalten diejenigen Messages, welche von der Middleware versendet werden und eine Statusänderung herbeiführen wollen den Typ «Controll-Command». *Weiterführende Informationen bezüglich dieses Nachrichten-Typs sind im Unterabschnitt 4.2.1.4 - Control-Command ausgeführt.*

Nachstehend findet sich eine Darstellungsübersicht der Nachrichten.

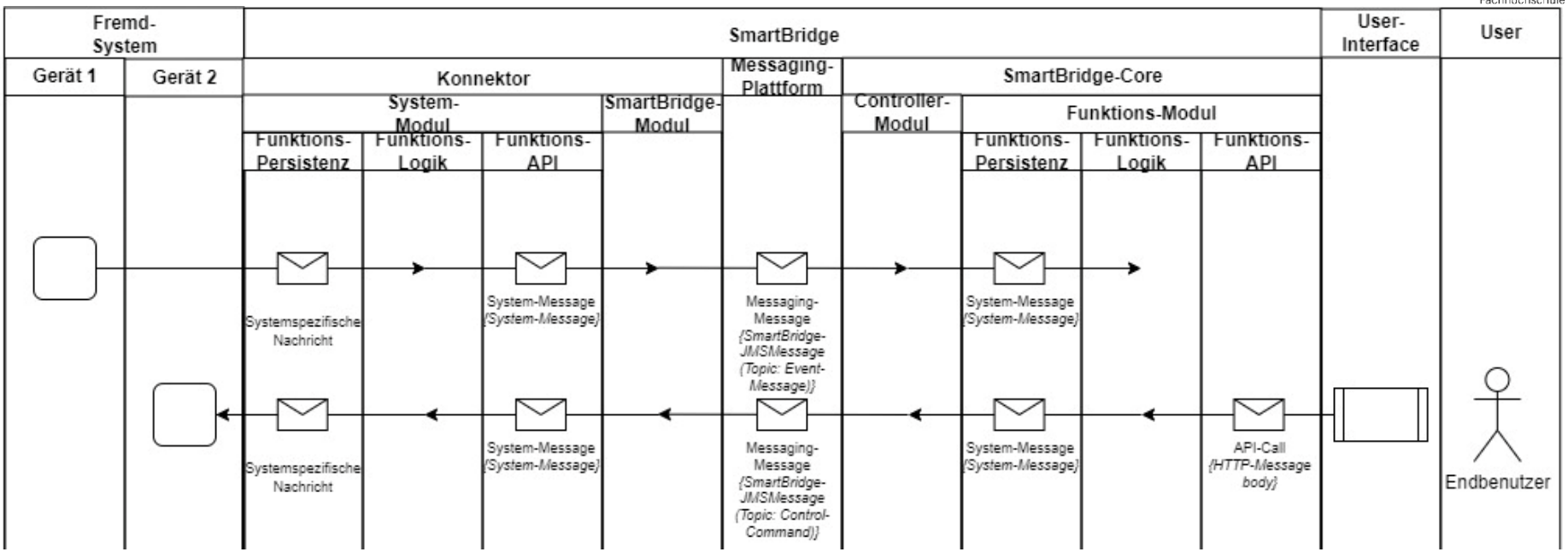


Abbildung 55: Darstellungsübersicht der Nachrichten

6.1.2. Detaillierte Aufschlüsselung der Laufzeit-Sicht

Die Aufschlüsselung der Laufzeit-Sicht wurde in die Richtungen, vom Fremd-System zu User-Interface und in die entgegengesetzte Richtung, unterteilt.

6.1.2.1. Nachrichtenfluss ausgelöst durch einen Fremd-System-Event

Verarbeitung des Zustands-Updates im Konnektor

Nach dem Eingang des Zustands-Update-Events, wird dieser vom System-Modul innerhalb des Konnektors in eine 4.2.3.2 - Third-Party-System-Message mit dem Nachrichten-Thema 4.2.1.5 - Event-Message, umgewandelt. Nachdem die Nachricht mit allen relevanten Informationen aus der System-Datenbank des Modules angereichert wurde, wird diese dann an alle interessierten Module des Konnektors übermittelt. So gelangt die Nachricht an das Modul SmartBridge.

Übermittlung des Zustands-Updates vom Konnektor an die Middleware-API

Das SmartBridge-Modul nimmt dann diese Nachricht und verpackt sie, nach dem Envelope-Wrapper-Pattern (Hohpe & Woolf , Envelope Wrapper, 2003) in eine «SmartBridgeJMSMessage» und veröffentlicht diese im Anschluss auf dem Publish-Subscribe-Channel (Hohpe & Woolf , Publish-Subscribe Channel, 2003) des Benutzers unter dem Topic des Fremd-Systems. Da sich die Middleware für alle Messaging-Topics aller Benutzer interessiert, wird ihr auch die vom Konnektor veröffentlichte Nachricht zugestellt.

Verarbeitung des Zustands-Updates in der Middleware

Die 4.2.4 - Messaging Message wird daraufhin aus dem «Envelope» ausgepackt und die System-Message, welche den Payload des Envelope-Wrapper-Patterns darstellt, innerhalb der Middleware veröffentlicht. Aus dieser Message entnehmen alle Funktions-Module immer nur jene Angabe, welche für sie von Interesse ist und speichern die Information in dessen Modul-Datenbank.

Übermittlung des Zzustands-Updates von der Middleware-API an das User-Interface

Sobald ein User-Interface eine Anfrage an die API eines Middleware-Modules stellt, werden die Daten aus der Datenbank abgefragt und an das User-Interface übermittelt. Hierbei werden die Daten an die jeweilige Repräsentation des Modules übergeben. Zu beachten ist hierbei, dass gewisse Felder wie die Datenbank-ID nicht an das User-Interface übergeben werden.

Mit zunehmender Repräsentationsgrössen der Objekte wäre ein Wechsel vom «Embedded Entity»-Pattern (Zimmermann et al., 2022) hin zum «Linked Information Holder»-Pattern (Zimmermann et al., 2022) in Betracht zu ziehen.

Serverseitig-basierende Übermittlung des Zustands-Updates an das User-Interface

Um bei Zustandsänderung, die basierend auf einer System-Message innerhalb der Middleware verteilt werden, das User-Interface über das Update zu informieren, wird in einem weiteren Schnitt das Etablieren von WebSockets umgesetzt.

WebSockets

WebSockets ermöglichen eine bi-direktionale Verbindung zwischen dem User-Interface und der Middleware. Die Übertragung selbst ist TCP-basiert was sicherstellt, dass die Informationen beim Gegenüber auch ankommen. Zudem geschieht die Übertragung selbst auf dem http-Protokoll und nutzt daher die ohnehin schon meist offenen Ports der Netzwerk-Komponenten. Daher eignen sich WebSokets ausgezeichnet für diesen Anwendungsfall.

Die Umsetzung der WebSockets-Verbindungen nach dem Anmelden auf dem User-Interface wird ausserhalb dieser Arbeit umgesetzt.

6.1.2.2. Nachrichtenfluss, ausgelöst durch einen Endbenutzer über ein User-Interface

Sobald ein Endbenutzer eine Zustands-Änderung an einem Fremd-System vornehmen möchte, so kann er dies über die ihm vom User-Interface zur Verfügung gestellte Option bewerkstelligen. Beim Demo-User-Interface wurde dies mittels Buttons auf den Geräte-Kacheln gelöst.

Übermittlung des Schalt-Befehles vom User-Interface an die Middleware-API
Diese für den Schaltbefehl benötigte Information wird im Body eins RESTful http Web API-Calls an die API des Command-Suite-Moduls der Middleware übermittelt.

Verarbeitung des Zustands-Updates in der Middleware
Sobald die eine entsprechender API-Call im Command-Suite-Modul eingeht, werden die entsprechenden Informationen aus dem API-Call entnommen und eine entsprechend System-Message wird erstellt. Diese wird dann innerhalb der Middleware an alle interessierten Module übermittelt, so auch an das Orchestration-Modul.

Übermittlung des Schalt-Befehles von der Middleware an den Konnektor
Das Orchestration-Modul verpackt die System-Message, nach der Evaluation an welchen Konnektor die Nachricht gesendet werden soll, in eine SmartBridgeJMSMessage und veröffentlicht diese wieder auf der Messaging-Plattform von ActiveMQ Artemis.

So wird das SmartBridge-Modul, stellvertretend für den gesamten Konnektor, über die gewünschte Statusänderung informiert.

Verarbeitung des Zustands-Updates im Konnektor
Das SmartBridge-Modul entpackt dann die Nachricht und versendet sie an die anderen interessierten Module.

Nachstehend wird der Nachrichtenfluss, ausgelöst durch einen Endbenutzer über ein User-Interface, grafisch dargestellt:

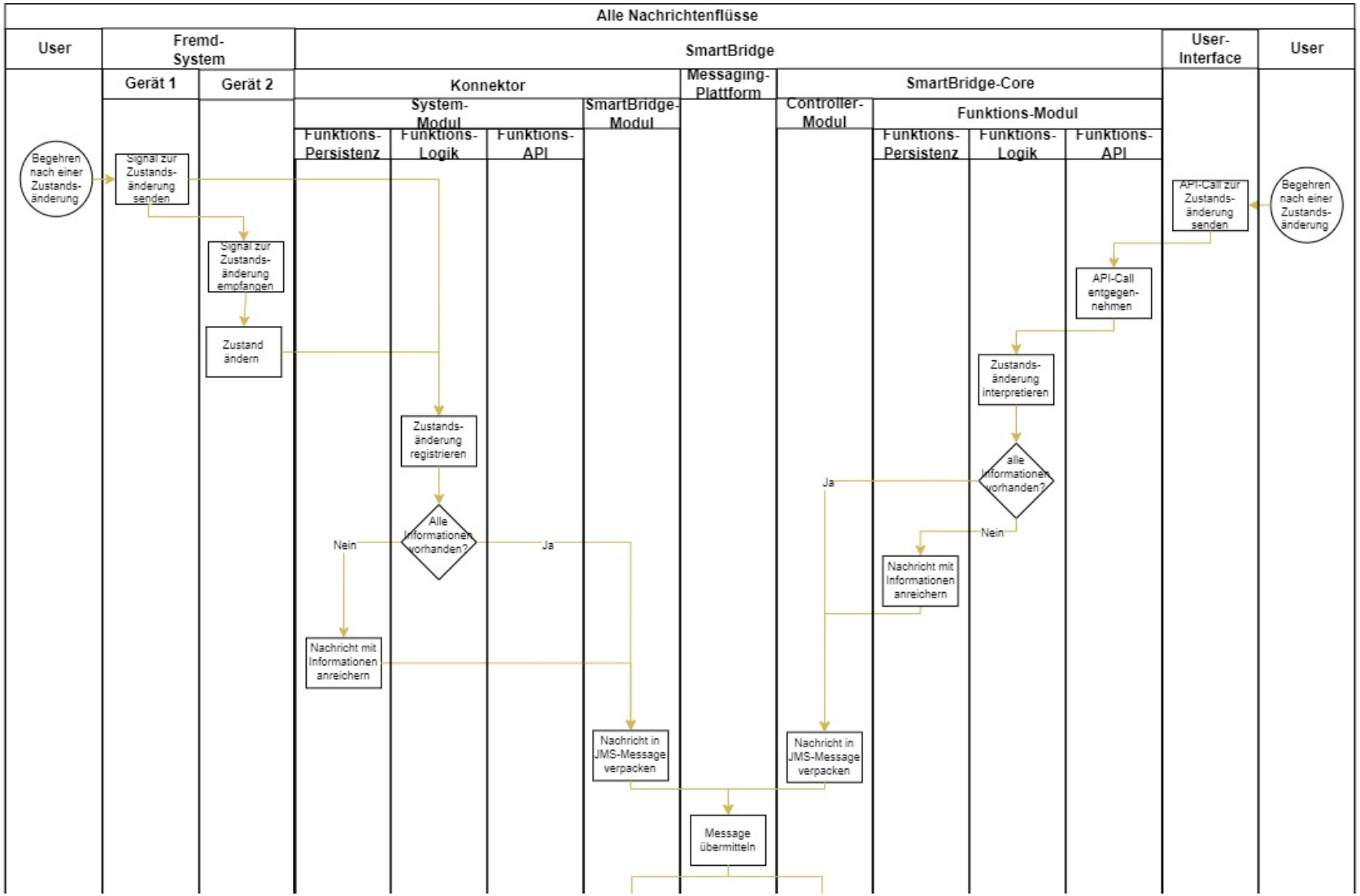


Abbildung 56: Nachrichtenfluss ausgelöst durch einen Endbenutzer über ein User-Interface - Teil 1

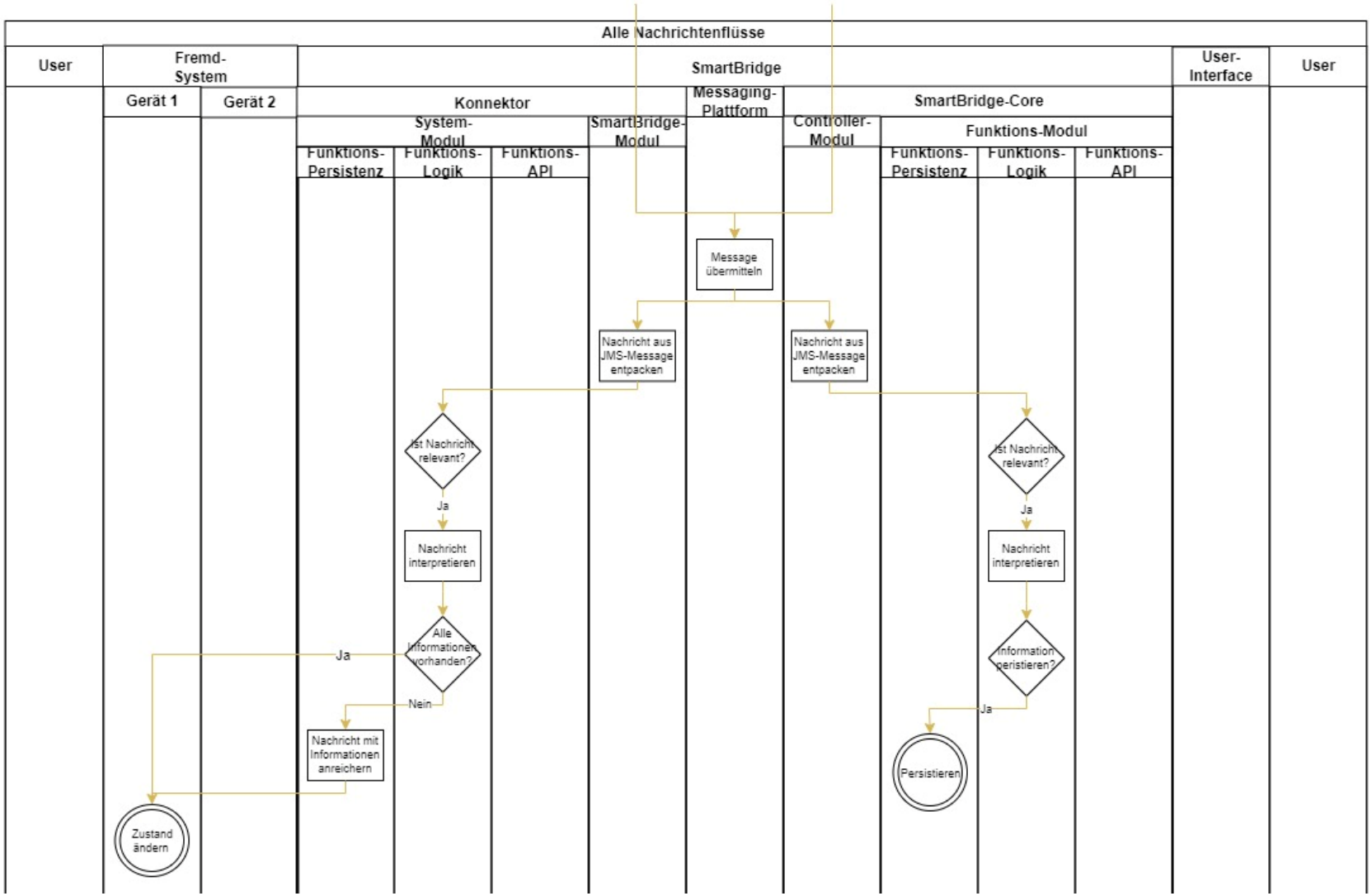


Abbildung 57: Nachrichtenfluss ausgelöst durch einen Endbenutzer über ein User-Interface - Teil 2

6.2. Prozesse

Um die Abläufe innerhalb der Konnektoren und der Middleware besser zu verstehen, wird in den folgenden Abschnitten auf ausgewählte Schlüsselprozesse eingegangen.

6.2.1. Verbindungsaufbau der Konnektoren

Eine der Herausforderungen bestand darin, eine Verbindung mit den Konnektoren aufbauen zu können, wenn diese erstmals gestartet werden. Um dies zu adressieren, verfügt die Middleware-API über einen Endpoint, welcher neben dem Namen des Konnektors und dessen Typ auch die User-ID benötigt. Basierend auf diesen Angaben wird ein neues Topic auf dem ActiveMQ Artemis Kanal des Users eröffnet. Die Antwort welche der Konnektor erhält verfügt sowohl über die Broker-URL als auch über die ID des Konnektors.

Untenstehend wird im Ablaufdiagramm das Hinzufügen eines neuen Konnektors erläutert:

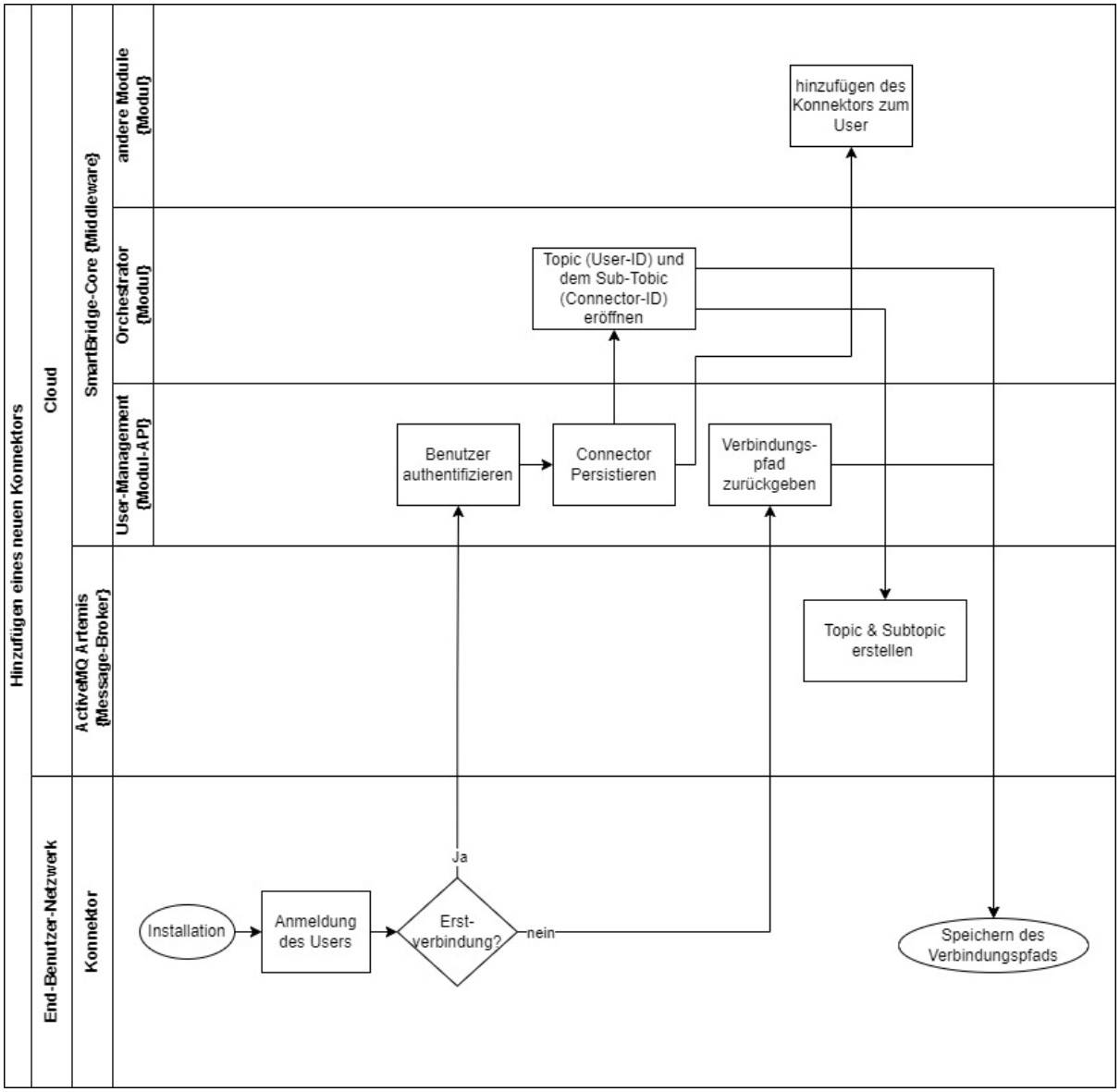


Abbildung 58: Hinzufügen eines neuen Konnektors

7. Verteilungssicht

Das Kapitel Verteilungssicht befasst sich mit den physikalischen Orten, an denen die verschiedenen Komponenten betrieben werden und zielt daher auf die Hersteller der SmartHome- und IoT-Systeme und die User-Interface Entwickler ab.

7.1. Deployment

Im folgenden Abschnitt wird untersucht, an welchem Standort die einzelnen Teilbereiche des SmartBridge-Ökosystems betrieben werden.

7.1.1. Azure-Region

Bei der Wahl der Azure-Region wurde explizit darauf geachtet, dass sich alle in der Cloud befindlichen Applikationen in derselben Azure-Region befinden. Dies aus dem Grund, dass die physikalische Limitation der Datenübertragung in Form von Latze zwischen den Applikationen so gering als möglich sind.

Bei der Wahl der Azure-Region musste jedoch eine Abwägung getroffen werden, wo diese sein soll. Ausschlaggebend hierbei waren zum einen die Durchschnittskosten und zum anderen die Latenz in die Azure-Region (ausgehend von Rapperswil-Jona). Leider sind die Azure-Regionen, welche sich in der Schweiz befinden und daher über eine geringe Latenz verfügen, relativ teuer. Ein guter Trade-off für diesen PoC stellte die Azure-Region North-Central US dar. Sie verfügt über eine durchschnittliche Latenz, die etwas über 100 Millisekunden liegt und zudem verfügt sie über die zweitgünstigsten Preise aller Azure-Regionen.

7.1.2. Azure Pipeline

Das Deployment der Middleware geschieht automatisiert durch ein Skript, welches sich auf der Azure DevOps Entwicklungsplattform befindet. Dieses Skript versucht in einem ersten Schritt die Applikation in ein ausführbares Programm zu bauen. Im Anschluss werden alle Tests ausgeführt und am Ende wird die fertig gebaute Applikation in einen Bereich kopiert, welcher für die Release-Pipeline zugänglich ist.

7.1.3. Azure Release Pipeline

Nachdem die Pipelines durchgelaufen sind, kann eine gebaute Applikations-Version veröffentlicht werden. Hierbei muss beim erstmaligen Aufsetzen die Ressource, welche veröffentlicht werden möchte, angeben werden.

7.1.4. Applikationen, die auf der Azure-Umgebung laufen

Die Middleware selbst besteht aus dem Middleware-Server, den Datenbanken der Module und der Messaging Plattform.

7.1.4.1. Middleware-Server

Der Middleware-Server, welcher den Namen «SmartBridge-Core» trägt, wird in einem App-Service betrieben. Betrieben wird sie als Web-Applikation, welches eine Ausprägung der App-Services auf Azure ist.

Beim Abo-Modell kam das «Basic (B1)» zum Einsatz, welches 1.75 GB Memory und einen virtuellen Prozessor-Kern umfasst. Dies kostet ca. 12 CHF jeden Monat.

7.1.4.2. Datenbanken

Die Datenbanken der Module, welche zusammen den Middleware-Server bilden, befinden sich auf demselben Datenbankserver. Dies stellt dieselbe Architektur wie beim Middleware-Server dar. Es

sind jeweils eigenständig lauffähige Applikation Bereiche, welche sich aktuell aber noch auf derselben Laufzeit-Instanz befinden.

Beim Datenbank-Server handelt es sich um einen «Azure Database for PostgreSQL flexible server». Darauf befinden sich neben den Postgres- und Azure-internen Datenbanken folgende Modul-Datenbanken:

- «recordkeeper»
- «policyhub»
- «common»
- «commandsuite»
- «monitoringsuite»

In Bezug auf das Abo-Modell wurde das «Standard_B1ms» Modell» welches 1 Virtuellen Prozessor-Kern und 2-GiB Memory bietet, verwendet. Da die Datenmengen bisher noch sehr überschaubar sind, genügt das geringste Speicher-Abonnement, welches 32 GiB umfasst. Zusammen kostet dieses Setup ca. 15 CHF pro Monat.

7.1.4.3. Messaging Plattform

Der ActiveMQ Artemis Broker wurde in einem Docker-Container als Azure Container instance Deployed.

Der Betrieb der Azure Container instance schlägt pro Monat mit ungefähr 30 CHF zu buche.

8. Querschnitt-Konzepte

Im Kapitel 8, liegt der Mehrwert für die Smart Home- und IoT-System Hersteller darin, Knowhow über mehrere Teilbereiche des SmartBridge-Ecosystems hinweg zu erfahren.

8.1.1. Lizenz-Abhängigkeiten

In diesem Unterabschnitt werden die Lizenzen der externen Abhängigkeiten behandelt.

8.1.1.1. Apache 2.0 Lizenz

Die Apache License 2.0 ist eine weit verbreitete Open-Source-Lizenz, die von der Apache Software Foundation entwickelt wurde. Sie ist darauf ausgelegt, ein hohes Mass an Freiheit und Flexibilität zu bieten, während sie gleichzeitig die Interessen der Rechte-Inhaber schützt. Die Apache License 2.0 ist eine der beliebtesten Lizenzen in der Open-Source-Community und wird von vielen Projekten verwendet.

Die Lizenz umfasst folgende Rechte und Pflichten:

- ***Kommerzielle Nutzung***
  Die Lizenz erlaubt es, die Software sowohl für private als auch für kommerzielle Zwecke zu verwenden.
- ***Verteilung***
  Nutzer dürfen die Software in unveränderter oder veränderter Form weitergeben.
- ***Modifikation***
  Der Quellcode kann nach Belieben modifiziert und weiterentwickelt werden.
- ***Unterlizenzierung***
  Es ist erlaubt, die Software unter zusätzlichen oder anderen Lizenzbedingungen weiterzugeben, solange die Bedingungen der Apache License 2.0 erfüllt bleiben.
- ***Lizenztext beifügen***
  Jede Kopie der Software, ob modifiziert oder nicht, muss eine Kopie der Apache License 2.0 enthalten.
- ***Hinweise auf Änderungen***
  Jede modifizierte Datei muss einen auffälligen Hinweis darauf enthalten, dass die Datei geändert wurde sowie eine Beschreibung der vorgenommenen Änderungen.
- ***Beibehaltung von Hinweisen***
  In der Quellform der Software müssen alle bestehenden Copyright-, Patent-, Marken- und Attribution-Hinweise beibehalten werden.
- ***Bereitstellung einer NOTICE-Datei***
  Wenn die Originalsoftware eine NOTICE-Datei enthält, müssen auch die abgeleiteten Werke eine lesbare Kopie dieser Datei enthalten, mit entsprechenden Anpassungen.

Beschränkungen der Lizenz:

- ***Lizenztext beifügen***
  Der volle Text der EPL 1.0 muss bei jeder Weitergabe der Software beigefügt werden um sicherzustellen, dass die Nutzer über ihre Rechte und Pflichten informiert sind.
- ***Bereitstellung des Quellcodes***
  Wenn modifizierte Versionen der Software weitergegeben werden, muss auch der Quellcode dieser Modifikationen verfügbar gemacht werden.

-

Abhängigkeiten dieses Projektes unter dieser Lizenz:

- Spring Boot Starter Parent
- Spring Boot Starter Artemis
- Spring Boot Starter Data JPA
- Spring Boot Starter HATEOAS
- Spring Boot Starter Web
- Spring Boot Starter Test
- Spring Modulith BOM
- Spring Modulith Starter Core
- Spring Modulith Starter JPA
- Spring Plugin Core
- Spring Modulith BOM
- Spring Boot Configuration Processor
- Spring Boot DevTools
- MapStruct
- Spring AOP
- Spring Boot Starter Cache
- SpringDoc OpenAPI Starter WebMVC UI
- Jackson Datatype JSR310
- Commons SunCalc
- JJWT API
- JJWT Impl
- JJWT Jackson
- Spring Security Core
- Spring Security Web
- Spring Security Config
- Spring WebSocket
- SpringFox Swagger2
- SpringFox Swagger UI
- javax.servlet-api

8.1.1.2. Eclipse Public Lizenz 1.0

Die Eclipse Public License 1.0 (EPL 1.0) ist eine Open-Source-Lizenz, der Eclipse Foundation. Sie ist ausgerichtet, sowohl die Freiheiten der Open-Source-Entwicklung zu fördern als auch die Interessen der Rechte-Inhaber zu schützen. Die EPL 1.0 ist besonders für Softwareprojekte geeignet, bei denen die Weitergabe von Änderungen und die Zusammenarbeit im Vordergrund stehen.

Die Lizenz umfasst folgende Rechte und Pflichten:

- ***Kommerzielle Nutzung***
  Die Lizenz erlaubt es, die Software sowohl privat als auch kommerziell zu nutzen.
- ***Verteilung***
  Entwickler dürfen die Software weitergeben, sowohl in unveränderter als auch in veränderter Form.
- ***Modifikation***
  Die Lizenz erlaubt es, den Quellcode zu verändern und weiterzuentwickeln.
- ***Weitergabe von Änderungen***
  Jede modifizierte Version der Software muss ebenfalls unter der EPL 1.0 weitergegeben werden. So müssen Änderungen am Quellcode, die veröffentlicht werden, auch frei verfügbar gemacht werden.
- ***Lizenztext beifügen***
  Der volle Text der EPL 1.0 muss bei jeder Weitergabe der Software beigefügt werden um sicherzustellen, dass die Nutzer über ihre Rechte und Pflichten informiert sind.
- ***Bereitstellung des Quellcodes***
  Wenn modifizierte Versionen der Software weitergegeben werden, muss auch der Quellcode dieser Modifikationen verfügbar gemacht werden.

Beschränkungen der Lizenz:

- ***Keine Haftung***
  Die Lizenz schliesst jegliche Haftung für Schäden aus, die durch die Nutzung der Software entstehen könnten.
- ***Keine Gewährleistung***
  Die Software wird ohne jegliche Gewährleistung bereitgestellt.

Abhängigkeiten dieses Projektes unter dieser Lizenz:

- JUnit

8.1.1.3. MIT Lizenz

Die MIT License ist eine der einfachsten und am weitesten verbreiteten Open-Source-Lizenzen. Sie ist dafür bekannt, sehr permissiv und unkompliziert zu sein, was sie zu einer beliebten Wahl für viele Entwickler und Open-Source-Projekte macht. Die MIT License bietet eine hohe Flexibilität und erlegt nur minimale Einschränkungen auf.

Die Lizenz umfasst folgende Rechte und Pflichten:

- ***Kommerzielle Nutzung***
  Die Lizenz erlaubt es, die Software sowohl privat als auch kommerziell zu nutzen.
- **Verteilung**
  Nutzer dürfen die Software in unveränderter oder veränderter Form weitergeben.
- ***Modifikation***
  Der Quellcode kann nach Belieben verändert und weiterentwickelt werden.
- ***Unterlizenzierung***
  Die Software kann unter zusätzlichen oder anderen Lizenzbedingungen weitergegeben werden.
- **Lizenztext beifügen**
  Der vollständige Lizenztext der MIT License muss in allen Kopien oder signifikanten Teilen der Software enthalten sein.

Beschränkungen der Lizenz:

- ***Keine Haftung***
  Die Lizenz schliesst jegliche Haftung für Schäden aus, die durch die Nutzung der Software entstehen könnten.
- ***Keine Gewährleistung***
  Die Software wird ohne jegliche Gewährleistung bereitgestellt.

Beschränkungen der Lizenz:

- ***Keine Haftung***
  Die Lizenz schliesst jegliche Haftung für Schäden aus, die durch die Nutzung der Software entstehen könnten.
- ***Keine Gewährleistung***
  Die Software wird ohne jegliche Gewährleistung bereitgestellt.

Abhängigkeiten dieses Projektes unter dieser Lizenz:

- Mockito JUnit Jupiter
- Mockito Core

8.1.1.4. PostgreSQL Lizenz

Die PostgreSQL License ist eine Open-Source-Lizenz, die spezifisch für das PostgreSQL-Datenbanksystem entwickelt wurde. Sie ist eine sehr permissive Lizenz, die der MIT License ähnelt und Entwicklern grosse Freiheiten bei der Nutzung und Modifikation der Software bietet. Die PostgreSQL License stellt sicher, dass die Software offen und für vielfältige Anwendungen zugänglich bleibt.

Die Lizenz umfasst folgende Rechte und Pflichten:

- ***Kommerzielle Nutzung***
  Die Lizenz erlaubt es, die Software sowohl privat als auch kommerziell zu nutzen.
- ***Verteilung***
  Nutzer dürfen die Software in unveränderter oder veränderter Form weitergeben.
- ***Modifikation***
  Der Quellcode kann nach Belieben verändert und weiterentwickelt werden.
-
- **Unterlizenzierung**
  Es ist erlaubt, die Software unter zusätzlichen oder anderen Lizenzbedingungen weiterzugeben.
- ***Lizenztext beifügen***
  Der vollständige Lizenztext der PostgreSQL License muss in allen Kopien oder signifikanten Teilen der Software enthalten sein. Dies stellt sicher, dass die Lizenzbedingungen transparent sind und jeder Nutzer über seine Rechte und Pflichten informiert ist.

Beschränkungen der Lizenz:

- ***Keine Haftung***
  Die Lizenz schliesst jegliche Haftung für Schäden aus, die durch die Nutzung der Software entstehen könnten.
- ***Keine Gewährleistung***
  Die Software wird ohne jegliche Gewährleistung bereitgestellt.

8.1.1.5. GNU General Public License

Die GNU General Public License (GNU GPL) ist ebenfalls eine sehr weitverbreitete Open-Source-Lizenz. Sie ist dafür bekannt, dass sie im Vergleich zu anderen Lizenzen strenger ist, da sie die Freiheit der Benutzer sicherstellt und sicherstellt, dass alle modifizierten Versionen der Software ebenfalls frei verfügbar sein müssen.

Die Lizenz umfasst folgende Rechte und Pflichten:

- ***Kommerzielle Nutzung***
  Die Lizenz erlaubt es, die Software sowohl privat als auch kommerziell zu nutzen, unter der Bedingung, dass die daraus abgeleiteten Werke ebenfalls unter der GNU GPL veröffentlicht werden.
- ***Verteilung***
  Nutzer dürfen die Software in unveränderter oder veränderter Form weitergeben, müssen jedoch den Quellcode und die Lizenz beifügen.
- ***Modifikation***
  Der Quellcode kann nach Belieben verändert und weiterentwickelt werden, jedoch müssen alle modifizierten Versionen ebenfalls unter der GNU GPL veröffentlicht werden.
- ***Unterlizenzierung***
  Die Software kann nicht unter zusätzlichen oder anderen Lizenzbedingungen weitergegeben werden. Alle abgeleiteten Werke müssen ebenfalls unter der GNU GPL stehen.
- ***Lizenztext beifügen***
  Der vollständige Lizenztext der GNU GPL muss in allen Kopien oder signifikanten Teilen der Software enthalten sein.

Beschränkungen der Lizenz:

- ***Keine Haftung***
  Die GNU-Lizenz schliesst jegliche Haftung für Schäden aus, die durch die Nutzung der Software entstehen könnten.
- ***Keine Gewährleistung***
  Die unter der GNU-Lizenz bereitgestellte Software wird ohne jegliche Gewährleistung bereitgestellt.

Abhängigkeiten dieses Projektes unter dieser Lizenz:

- Calimero

8.1.2. Bekannte Vulnerability

Neben den Lizenzen, unter welche die verwendeten externen Abhängigkeiten stehen, wurden auch dessen bekannte Schwachstellen untersucht. Bereits bei der Entwicklung wurde darauf geachtet, dass so wenig Abhängigkeiten mit bekannten Schwachstellen wie möglich verwendet wurden. Bei der letzten Überprüfung (07.06.2024 – 18:45) der externen Abhängigkeiten in Bezug auf Sicherheits-bedenken konnten bis auf die folgenden Sicherheitslücken keine weiteren festgestellt werden.

Sicherheitslücke	Problem	Betroffener Code	Betroffene Abhängigkeit
Cx78f40514-81ff / CWE-674	Unkontrollierte Rekursion	Apache Commons Collections	org.springframework.boot - spring-boot-starter-artemis - Version 3.3.0 (May 23, 2024)
CVE-2024-22259	Unzureichende URL-Validierung	UriComponentsBuilder	org.springframework.boot - spring-boot-starter-hateoas - Version 3.3.0 (May 23, 2024)

Tabelle 15: Übersicht bekannter Vulnerabilitäten

8.1.2.1. Sicherheitslücke CWE-674

Der CWE-674 bezeichnet eine Sicherheitslücke die auftritt, wenn rekursive Funktionen oder Datenstrukturen keine ordnungsgemässen Abbruchbedingungen haben. Dies führt zu unendlicher Rekursion, die Stack-Overflows verursacht und die Anwendung abstürzen lässt. Diese kann für einen Denial-of-Service (DoS)-ähnlichen Angriff missbraucht werden.

Diese Sicherheitslücke ist insofern besonders kritisch für dieses Projekt, da die betroffene externe Abhängigkeit für die Datenübertragung zwischen den Konnektoren und der Middleware zuständig ist. Daher betrifft diese Sicherheitslücke eine der wenigen kritischen Stellen dieser Architektur. Zudem ist diese externe Abhängigkeit in allen Konnektoren als auch in der Middleware und somit einem Grossteil des Projektes vorhanden.

8.1.2.2. Sicherheitslücke CVE-2024-22259

Der CVE-2024-22259 beschreibt eine Sicherheitslücke im Spring Framework, die sich aus der Verarbeitung und Validierung von URLs ergibt. Diese Schwachstelle kann von Angreifern ausgenutzt werden, um Open Redirect- und Server-Side Request Forgery (SSRF)-Angriffe durchzuführen.

Open Redirect-Angriff:
Benutzer können auf bösartige Websites umgeleitet werden, was zu Phishing-Angriffen führen kann.

SSRF-Angriff:
Der Server kann dazu gebracht werden, Anfragen an unerwünschte Ziele zu senden, was zu unbefugtem Zugriff auf interne Ressourcen führen kann.

Dies ist insofern kritisch für das Projekt, dass durch einen erfolgreichen Angriff bei einem Endbenutzer das Vertrauen in die Middleware verloren ist. Da diese Sicherheitslücke, welche während dieser Arbeit entdeckt und veröffentlicht wurde, bis zur letzten Überprüfung (07.06.2024 – 18:45) noch nicht durch

eine neuere Version geschlossen werden konnte, bleibt dies eine bestehende Sicherheitslücke der Middleware.

8.2. Lizenz dieser Arbeit

Für diese Arbeit, welche sich in der Proof-of-Concept Phase befindet und deren Code nicht veröffentlich werden wird, greifen die Mechanismen in Bezug auf die Lizenzierung der GNU GPL noch nicht. Diese würden erst bei einer Veröffentlichung oder ab der Weitergabe an Dritte in Kraft treten.

Für den weiteren Verlauf dieses Projektes, welches über diese Arbeit hinaus geht, bedeutet dies, dass eine Ablösung der Calimero-Dependency zwingend sein wird, um das Projekt unter einer anderen Lizenz als der GNU GPL veröffentlichen zu können. Im Rahmen dieser Arbeit, welche mit dem PoC die technische Machbarkeit evaluieren sollte, war es die effizienteste Lösung die Calimero-Dependency zu nutzen. Dies trotz der Tatsache, dass eine zukünftige Ablösung anstehen wird.

Im Rahmen dieser Arbeit wurde zudem auch evaluiert, unter welche Lizenz das Projekt bei einer Veröffentlichung gestellt werden wird. Unter der Voraussetzung, dass die Calimero-Dependency zuvor noch abgelöst werden wird, wurde die Apache License 2.0 als gesamthaft passendste Lizenz evaluiert. Wichtig war der Schutz der Rechte als Entwickler und die Flexibilität, die diese Lizenz bietet.

Hauptvorteile der Apache License 2.0 in Bezug auf dieses Projekt:

- **Haftungsausschluss**
  Die Apache Lizenz bietet einen Haftungsausschluss der sicherstellt, dass ich als Entwickler nicht für mögliche Schäden haftbar gemacht werden kann, die durch die Nutzung der Software entstehen könnten. Dies ist vor allem wichtig, da es sich bei den Smart Home- und IoT-Systemen für den Endbenutzer intime Systeme handeln kann.

- **Weiterentwicklung**
  Die Apache Lizenz erlaubt es, den Quellcode nach Belieben zu modifizieren und weiterzuentwickeln. Diese Flexibilität ist entscheidend in der PoC-Phase, in der häufige Änderungen und Anpassungen notwendig sind, um verschiedene Ansätze zu testen und die beste Lösung zu finden.

- **Open-Source-Community**
  Die Apache License 2.0 ist eine der am weitesten verbreiteten und akzeptierten Open-Source-Lizenzen weltweit. Diese breite Akzeptanz schafft Vertrauen bei potenziellen Nutzern und Mitwirkenden, da sie mit den Bedingungen und Vorteilen dieser Lizenz vertraut sind.

Die Wahl der Apache 2.0 Lizenz für diese Arbeit ist eine strategische Entscheidung, die sowohl den aktuellen Entwicklungsbedarf als auch die zukünftigen Möglichkeiten berücksichtigt.

9. Design-Entscheidungen

Da die Entwurfsentscheidungen Einfluss auf alle Bereiche haben, richtet sich dieses Kapitel an alle Zielgruppen. Für die Endbenutzer die diese Anwendung nur benutzen wollen , ist diese Kapitel jedoch nicht von Zentraler Bedeutung.

9.1. Architecture Decision Records

Alle Entwurfsentscheidungen werden in sogenannten architecture decision records (ADR's) festgehalten. Diese wurden basierend auf der Vorlage des Y-Template der ABB Software Improvment Initiative getätigt.

9.1.1. Monolith versus Micro-Services versus Modularer Monolith (Modulith)

Im Zusammenhang mit der potenziell grossen Anzahl an Anfragen, welche ein solcher Anwendungsfall mit sich bringen kann,
- ***begegneten wir*** der Herausforderung, zu evaluieren welche Bereiche der Middleware wie zu skalieren ist, daher
- ***beschlossen wir*** den Einsatz eines Modulareren Monolithen (Modulithische Architektur)
- ***und gegen*** eine Monolithische- und auch gegen eine Micro-Service-Architektur
- ***um zu erreichen,*** dass nur eine Instanz deployed und betrieben werden muss und später noch immer einzelne Module herausgelöst und eigenständig betrieben werden können
- ***und in Kauf zu nehmen***, dass die aktuelle Architektur wahrscheinlich nicht die endgültige Architektur ist.

9.1.2. Architektur der Module

Im Zusammenhang mit Wahl der Architektur für die Modulith-Module
- ***begegneten wir*** der Herausforderung, die dafür passende Architektur zu evaluieren, daher
- ***beschlossen wir*** uns für den Einsatz einer Drei-Schichten-Architektur
- ***und gegen*** den Einsatz einer Onion-Architektur
- ***um zu erreichen,*** dass die anfängliche Komplexität nicht den Rahmen dieser Arbeit sprengt
- ***und in Kauf zu nehmen***, dass die Verwaltung der Datenkonsistenz über mehrere Datenbanken hinweg komplexer sein kann.

9.1.3. Apache Kafka versus Apache Active MQ

Grundvoraussetzungen an die Message-Broker

- Unterstützung von Spring
- Vorhandene Produkte auf dem Azure Marktplatz
- Skalierbarkeit während dem Betrieb

Im Zusammenhang mit dem Message Broker
- ***begegneten wir*** der Herausforderung eine grosse Anzahl an Anfragen bewältigen zu müssen, daher
- ***beschlossen wir***, uns für den Einsatz von ActiveMQ
- ***und gegen*** den Einsatz von Apache Kafka
- ***um zu erreichen,*** dass das für den Arbeitsumfang passendste Produkt gewählt wird
- ***und in Kauf zu nehmen***, dass die Anzahl der Anfragen während des PoC's nicht 50'000 Messages pro Sekunde übersteigen sollten.

9.1.4. Ein Messaging-Kanal pro User

Im Zusammenhang mit dem eingesetzten Message Broker
- ***begegneten wir*** der Wahl des Enterprise Integration Patterns für die Übermittlung der Nachrichten, dabei
- ***beschlossen wir***, uns für den Einsatz von einem Kanal pro Benutzer, welcher im Public-Subscribe Modus betrieben wird und jeder Konnektor ein eigenes Topic im Kanal bekommt
- ***und gegen*** den Einsatz eines eigenen Point-to-Point Kanales pro Benutzer
- ***um zu erreichen,*** dass die Anzahl der Kanäle überschaubarer bleibt
- ***und in Kauf zu nehmen***, dass die anderen Konnektoren die Kommunikation mitbekommen können.

9.1.5. Java 17 LTS anstelle Java 21 betreffend Support auf Azure

Im Zusammenhang mit der Wahl (zu Beginn des Projekts) der zu verwendenden Java-Version
- ***begegneten wir*** der Herausforderung einer noch nicht umfassenden Unterstützung der Java-Version 21 auf den Microsoft-Azure-Plattformen, daher
- ***beschlossen wir***, auf die Java-Version 17 zu gehen
- ***und gegen*** die neueste LTS Java-Version 21
- ***um zu erreichen,*** dass es zu unerwarteten Verzögerungen basierend auf fehlendem Support kommt
- ***und in Kauf zu nehmen***, dass ein späteres Upgrade kleine funktionale Anpassung mit sich bringen können.

9.1.6. Keine serverseitige Zertifikatsüberprüfung bei Shelly-Geräten

Im Zusammenhang mit der Verbindung zwischen dem Shelly-Konnektor und den Shelly-Geräten
- ***begegneten wir*** der Möglichkeit der serverseitigen Zertifikatsüberprüfung des Client-Zertifikates, dabei
- ***beschlossen wir***, uns aktiv darauf zu verzichten
- ***und gegen*** eine Shelly-geräteseitige Zertifikatsüberprüfung
- ***um zu erreichen,*** dass die anfängliche Komplexität nicht den Rahmen dieser Arbeit sprengt
- ***und in Kauf zu nehmen***, dass dies vor einer Veröffentlichung noch angepasst werden sollte.

9.1.7. Callback-Modul

Im Zusammenhang mit der von Sonos benötigten «Secure Callback URL», für das Benachrichtigen bei Statusänderungen des Sonos-Systems
- ***begegneten wir*** der Herausforderung, dass diese nicht innerhalb des Heim-Netzwerks sein darf, so
- ***beschlossen wir***, diese Arbeit auf den architektonischen Teil der Umsetzung zu beschränken
- ***und gegen*** die Statusänderungen im Rahmen dieser Arbeit zu implementieren
- ***um zu erreichen,*** dass die Komplexität nicht den Rahmen dieser Arbeit sprengt
- ***und in Kauf zu nehmen***, dass Änderungen, die manuell an den Playern oder über das Sonos-App vorgenommen werden, nicht an die Middleware übermittelt werden.

9.1.8. Ein Datenbankserver für die Relationalen Datenbanken der Middleware

Im Zusammenhang mit der Datenbank-Architektur
- ***begegneten wir*** der Entscheidung wie viele Datenbanken wir betreiben sollen und so
- ***beschlossen wir***, uns für verschiedene Datenbanken auf einem Datenbank-Server
- ***und gegen*** einen Datenbank-Server pro Datenbank
- ***um zu erreichen,*** dass wir kostenbewusst mit den Ressourcen auf der Azure-Umgebung umgehen
- ***und in Kauf zu nehmen***, dass bei einem Ausfall des Datenbank-Servers keine der Datenbanken mehr erreichbar sind.

9.1.9. Sonos-Konnektor innerhalb desselben Netzwerkes wie die Sonos-Speaker

Im Zusammenhang mit dem Betriebs-Standort des Sonos-Konnektors
- ***begegneten wir*** der Entscheidung, ob dieser als Cloud-Konnektor ungesetzt werden soll. Dabei
- ***beschlossen wir***, uns für die Umsetzung als klassischen In-Netzwerk-Konnektor
- ***und gegen*** eine Cloud-Konnektor-Variante
- ***um zu erreichen,*** dass die Zeit-Ressourcen des Arbeit-Umfanges angepasst eingesetzt werden
- ***und in Kauf zu nehmen***, dass dieser PoC den möglichen Betrieb von Cloud-Konnektoren nicht abdeckt.

9.1.10. Calimero als KNX-Kommunikations-Bibliothek

Im Zusammenhang mit der Kommunikation von KNX-Komponenten und deren Konnektor
- ***begegneten wir*** der Entscheidung, ob eine bestehende Java-Library eingesetzt werden soll. Hierbei
- ***beschlossen wir*** uns für den Einsatz des Projekts Calimero der Technischen Universität Wien
- ***und gegen*** das Erstellen einer eigenen Bibliothek zur Kommunikation mit den Komponenten
- ***um zu erreichen,*** dass der Einsatz der zeitlichen Ressourcen dem Umfang und dem Charakter der Arbeit als PoC entsprechend eingesetzt werden
- ***und in Kauf zu nehmen***, dass vor der Veröffentlichung der Middleware als kommerzielles Produkt die eingesetzte Java-Library (Calimero) abgelöst werden muss.

9.1.11. Daten-Hoheit der allgemeinen Daten-Attribute

Im Zusammenhang mit der Daten-Hoheit über die allgemeinen Daten wie User- Connector und Device
- ***begegneten wir*** der Entscheidung, dass genau ein Modul verantwortlich ist, dabei
- ***beschlossen wir***, uns dafür das Modul «User-Management» als verantwortlich zu nominieren
- ***und gegen*** die Möglichkeit der gemeinsamen und verteilten Verantwortung
- ***um zu erreichen,*** dass zu jeder Zeit, das «Single Source of Truth»-Prinzip sichergestellt werden kann
- ***und in Kauf zu nehmen***, dass das User-Management-Modul scheinbar modul-fremde Informationen halten muss.

9.1.12. Wahl der Anzahl Verbindungs-Module zwischen Konnektoren und Middleware

Im Zusammenhang mit der Architektur der Konnektoren
- ***begegneten wir*** der Entscheidung, ob jeder Konnektor ein Verbindungs-Modul haben soll oder ob sich pro Netzwerk, in welchem sich mindestens ein Konnektor befindet, nur ein Verbindungs-Konnektor befinden muss. Dabei
- ***beschlossen wir*** uns dafür, dass die Konnektoren eigenständig lauffähig sein sollen und somit dafür, dass jeder Konnektor ein eigenes Verbindungs-Modul hat
- ***und gegen*** die Möglichkeit, dass sich mehrere Konnektoren diese Instanz teilen
- ***um zu erreichen,*** dass jeder Konnektor eigenständig läuft und kein Single Point of Failure zu haben
- ***und in Kauf zu nehmen***, dass Konnektoren noch schlanker gebaut werden könnten.

9.1.13. Evaluation der System-Datenbank im Konnektor

Im Zusammenhang mit der Wahl der System-Datenbank im Konnektor
- ***begegneten wir*** der Wahl zwischen der Verwendung von SQLite- oder H2-Datenbanken
- ***beschlossen wir*** uns für die H2-Datenbanken
- ***und gegen*** die noch leichtgewichtigeren SQL-Lite-Datenbanken
- ***um zu erreichen,*** dass weitere Konnektoren auch die Möglichkeit zu Stored Proceduren und das Persistieren von Geospatial Data zur Verfügung steht
- ***und in Kauf zu nehmen***, dass 2 unterschiedliche Datenbanken pro Konnektor betrieben werden müssen.

9.1.14. Physischer Standort der System-Datenbank

Im Zusammenhang mit der Wahl des physischen Standortes der System-Datenbank
- ***begegneten wir*** der Entscheidung, diese in der Cloud oder beim Endbenutzer zu betreiben. Dabei
- ***beschlossen wir***, uns für den lokalen Betrieb in der Laufzeitumgebung des Konnektors
- ***und gegen*** einen Cloud-basierten Betrieb
- ***um zu erreichen,*** dass nur der Endbenutzer selbst Zugriff auf die Daten hat und die Latenz so gering wie möglich ist
- ***und in Kauf zu nehmen***, für den Abgleich neu hinzugefügter System-Komponenten mit der Middleware, mit zusätzlicher Kommunikation zu rechnen ist.

9.1.15. Sicherheit der Datenübertragung zischen Konnektor und Middleware

Im Zusammenhang mit der Übertragungsarchitektur
- ***begegneten wir*** der Entscheidung, ob die Informationen, welche zwischen Middleware und Konnektoren übertragen werden, zu verschlüsseln sind. In diesem Zusammenhang
- ***beschlossen wir***, uns dazu aktuell keine Verschlüsselungs-Mechanismen einzubetten
- ***um zu erreichen,*** dass das zur Verfügung stehende Zeit-Budget für die funktionale Evaluation der Durchführbarkeit einer solchen Middleware eingesetzt wird
- ***und in Kauf zu nehmen***, dass die aktuell übertragenen Informationen nicht den branchenüblichen Sicherheitsanforderungen standhalten.

9.1.16. Wahl der System-Datenbank für die Konnektoren

Im Zusammenhang mit dem Persistieren der fremd-system-spezifischen Daten
- ***begegneten wir***, der Entscheidung welche Datenbank diesbezüglich eingesetzt werden soll
- ***beschlossen wir***, uns für den Einsatz der H2-Datenbank im filebasierten Betriebsmodus
- ***und gegen*** den Einsatz von PostgreSQL-Datenbanken
- ***um zu erreichen,*** dass der Ressourcenverbrauch in einem angemessenen Verhältnis zu den zu speichernden Daten steht
- ***und in Kauf zu nehmen***, dass eine weniger verbreitete Datenbank eingesetzt wird.

9.1.17. MapDB anstelle von H2 für das Persistieren von Konfigurations-Parametern

Im Zusammenhang mit der Architektur der Konnektoren
- ***begegneten wir***, der Entscheidung welche Datenbank für das Persistieren der Konfigurations-Parameter eingesetzt werden soll. In diesem Zusammenhang
- ***beschlossen wir***, uns für den Einsatz der MapDB
- ***und gegen*** die bereits in einem anderen Modul verwendete H2-Datenbank
- ***um zu erreichen,*** dass die Daten in einer Map-Datenstruktur persistiert werden können und keine Relationale Tabelle für den Einsatz entfremdet werden muss und zudem das «Separation of Concerns»-Prinzip angewandt wird
- ***und in Kauf zu nehmen***, dass keine komplexen Abfragen der Datenbank gemacht werden können.

9.1.18. Versionierung der Messaging-Nachrichten

Im Zusammenhang mit der Messaging-Architektur
- ***begegneten wir*** der Herausforderung, dass die Konnektoren unabhängig von der Middleware upgedatet werden können und somit eine Rückwärtskompatibilität sichergestellt werden soll. Hierbei
- ***beschlossen wir*** uns für den Einsatz eines Versions-Feldes auf der Nachricht
- ***und gegen*** den Updatezwang von Konnektoren
- ***um zu erreichen,*** dass die Middleware auch noch mit Konnektoren kommunizieren kann, wenn diese jeweils auf unterschiedlichen Versionen sind. Zudem sollen damit trotzdem neue Features, die weitere Felder auf den Nachrichten benötigen, implementiert werden können
- ***und in Kauf zu nehmen***, dass ein zusätzliches Feld auf der Message übertragen werden muss.

9.1.19. Wahl des ActiveMQ Artemis Protokolls

Im Zusammenhang mit der Wahl des Protokolls für ActiveMQ Artemis während des PoC's
- ***begegneten wir***, der Entscheidung, ob das Core-Protokoll AMQP oder MQTT für die Kommunikation geeigneter ist. Hierbei
- ***beschlossen wir***, das Core-Protokoll einzusetzen
- ***und gegen*** die AMQP- oder MQTT-Protokolle
- ***um zu erreichen,*** dass die Implementierung der Funktionen im Vordergrund steht und die verfügbare Zeit nicht für das Einarbeiten in neue Protokolle verwendet werden muss
- ***und in Kauf zu nehmen***, dass vor einer Veröffentlichung, diese Entscheidung nochmals überarbeitet werden sollte und bei einem erfolgreichen Abschluss der PoC-Phase ein Wechsel auf das AMQP-Protokoll wahrscheinlich ist.

9.1.20. Übertragung der User-ID an die Middleware-API

Im Zusammenhang mit der Selektion der angefragten Daten bei der Middleware-API
- ***begegneten wir***, der Entscheidung, wie die User-ID an die Middleware übermittelt werden soll. Dabei
- ***beschlossen wir***, diese als Payload beim JWT-Token mitzuliefern
- ***und gegen*** das Übertragen im HTTP-Body der Anfrage
- ***um zu erreichen,*** dass sichergestellt werden kann, dass es sich wirklich um den User handelt
- ***und in Kauf zu nehmen***, dass bei jeder Anfrage das Token ausgelesen werden muss.

9.1.21. Entscheid für Spring-Security und gegen Identity und Access Management Provider

Im Zusammenhang mit der Umsetzung des «Identity & Access Managements»
- ***begegneten wir***, der Entscheidung dies selbst umzusetzen oder einen Provider zu nutzen. Dabei
- ***beschlossen wir***, uns für die eigene Umsetzung mittels Spring-Security
- ***und gegen*** einen Identity und Access Management Provider wie Auth0
- ***um zu erreichen,*** dass nur die für den PoC notwendigen Funktionalitäten umgesetzt werden müssen
- ***und in Kauf zu nehmen***, dass vor einer Veröffentlichung diese Entscheidung revidiert werden kann.

10. Qualitätsanforderungen

Dieses Kapitel ist sowohl für die Hersteller von Smart Home und IoT-Systemen als auch für die Entwickler von User-Interfaces von Interesse. Für die Endbenutzer ist dieses Kapitel nur insofern von Interesse, sofern sie sich für die Hintergründe und die Grundlagen, unter welchen ihr Produkt entwickelt wurde, interessieren.

10.1.1. C4-Komponenten-spezifische Qualitätsziele

Zusätzlich zu den 1.6.1 - Allgemeine Qualitätsziele aus dem Kapitel 1 - Einleitung des Dokumentes wurden die folgenden Qualitätsziele für einzelne Komponenten des SmartBridge-Ecosystems, wie Konnektoren oder die Middleware, zusätzlich definiert.

10.1.1.1. effizient (efficient)

Spezifikation nach arc42 Quality Model:
„ Perform functions within specified time, capacity and throughput parameters, using appropriate resources (like memory, network bandwith, threads)“ (Starke, arc42 Quality Model, 2022)

Da die Übertragung der Daten jeweils von allen Konnektoren auf die Server-Instanz über ein Messaging funktioniert ist es hierbei entscheidend, dass die Übermittlung der Messages sehr zeitnah und ohne erneutes Senden aufgrund eines Nachrichtenverlustes funktioniert.

10.1.1.2. sicher (save)

Spezifikation nach arc42 Quality Model:
„ Protect information and data so that persons or other products have only access to an extend appropriate to their types and levels, and to defend against attack patterns by malicious actors“ (Starke, arc42 Quality Model, 2022)

Die Daten, welche zu einem Benutzer gehören, sind vor unberechtigtem Zugriff zu schützen. So darf jeder Anwender nur diejenigen Daten sehen, die einem selbst zugeordnet werden können.

10.1.1.3. geeignet (suitable)

Spezifikation nach arc42 Quality Model:
„An abstract property, applicable to various objects. Provide properties that meet stated and implied needs of intended stakeholders.“ (Starke, arc42 Quality Model, 2022)

Das Deployment der Middleware ist der aktuellen Last-Anforderungen anzupassen. Zudem ist es entscheidend das Deployment ohne grosse Entwicklungs-Arbeiten an neue Last-Anforderungen anzupassen, um sowohl auf der horizontalen als auch der vertikalen Ebene skalieren zu können.

10.1.2. Qualitätsziele nach C4-Komponenten und Komponenten-Gruppen

Die zuvor ausgeführten Qualitätsziele werden nun auf die C4-Komponenten angewandt.

10.1.2.1. Middleware-API

Die Stakeholder der Middleware-API sind nicht die Endbenutzer, sondern sind Mobile-App und Web-Applikations-Entwickler, die ein eigenes User-Interface für die Middleware schreiben wollen. Daher werden nachfolgend die zur Erreichung der positiven User-Experience wichtigen Qualitätsziele für die Stakeholder genau ausspezifiziert. Als Grundlage dient das arc42 Quality Model.

zuverlässig (reliable)
Hauptmerkmale, welche Stakeholder unter zuverlässig verstehen nach dem arc42 Quality Model:

„(potential) Expectation for reliable of Developers:
- *reliably add new features or functions to the system without unwanted sideeffects.*
- *realiably predict the effects of changes to the system“ (Starke, #reliable, 2022)*

Auf die Middleware-API übertragen bedeutet dies, dass grossen Wert auf eine saubere Dokumentation der Funktionalitäten und dessen Verhalten gelegt wird. Zudem soll der Isolations-Strategie Priorität eingeräumt werden, so dass allfällige Fehler auf die ursprüngliche Fehlerquelle beschränkt bleiben.

brauchbar (usable)
Was Stakeholder unter brauchbar verstehen nach dem arc42 Quality Model:

„(potential) Expectation for usable of Developers:
- *understandable source-code and dependencies*
- *an appropriate technology-stack*
- *no (or at least predictable) side-effects when changing the system“*

(Starke, #usable, 2022)

Im Kontext der Middleware-API wird bei diesem Qualitätsziel darauf geachtet, einen branchen-üblichen API-Beschreibungsstandard zu verwenden. Zudem soll bei der Auswahl der Art der API grosser Nachdruck auf eine breite Palette an verschiedenen User-Interface-üblichen Technologien gelegt werden.

flexibel (flexible)
Das arc42 Quality Model hat hierfür keine Definition, was Entwickler unter flexibel verstehen.

In Bezug auf die Flexibilität soll darauf geachtet werden, dass Änderungen auch rückwärts-kompatibel sind und Abfragen mittels Filter vom User-Interface Entwickler auf seinen Anwendungsfall spezifisch angepasst werden können, ohne einen individuellen Endpunkt beantragen zu müssen.

operable (bedienbar)
Was Stakeholder unter bedienbar verstehen nach dem arc42 Quality Model:

„(potential) Expectation for operable of Developers:
- *automated test and build*
- *appropriate automation of deployments*
- *appropriate similarity of development and production environments “*

(Starke, #operable, 2022)

Für die Zukunft, welche über diese Arbeit hinaus geht, soll es eine Test-Umgebung geben in welcher die User-Interface Entwickler alle Endpoints, basierend auf einem Testdaten-Set, testen können.

10.1.2.2. Middleware

Beim Herzstück der Gesamtlösung, der Middleware selbst, ist die Zielgruppe das DevOps-Team, welches die Middleware wartet, weiterentwickelt und betreibt. Daher soll bei der Architektur und der Erstellung dieser auf die Bedürfnisse dieser Zielgruppe vermehrt geachtet werden.

zuverlässig (reliable)
Was Stakeholder unter zuverlässig verstehen nach dem arc42 Quality Model:

„(potential) Expectation for reliable of Developers:
- *reliably add new features or functions to the system without unwanted side-effects.*
- *realiably predict the effects of changes to the system“ (Starke, #reliable, 2022)*

„(potential) Expectation for reliable of Admin:
- *reliably start, configure and monitor the system*
- *reliable release and update processes“ (Starke, #reliable, 2022)*

Durch den Einsatz von Entwurfsprinzipien und Patterns sollen Fehler weit möglichst verhindert und ihre Auswirkungen eigedämmt werden.

Ausserdem soll es ein übergreifendes Log-System geben, welches es ermöglicht, technische Fehlerursachen schnell zu finden und zu beheben.

brauchbar (usable)
Was Stakeholder unter brauchbar verstehen nach dem arc42 Quality Model:

„(potential) Expectation for usable of Developers:
- *understandable source-code and dependencies*
- *an appropriate technology-stack*
- *no (or at least predictable) side-effects when changing the system“*

(Starke, #usable, 2022)

„(potential) Expectation for usable of Admin:
- *easy to deploy and install,*
- *installability*
- *deployability “*

(Starke, #usable, 2022)

Durch die Verwendung von ausgewählten Namen und entsprechender Code-Dokumentation sollen sich Entwickler mit einiger Entwicklungserfahrung selbständig in den Code einarbeiten können.

Das Deployment soll automatisch durch eine Pipeline erfolgen.

flexibel (flexible)
Dass arc42 Quality Model hat hierfür keine Definition was Entwickler und Administratoren unter flexibel verstehen.

Für die Entwicklung der Middleware soll auf eine sinnvolle funktionale Modularisierung geachtet werden. So sollen flexibel Anpassungen vorgenommen und Erweiterungen an der Business-Logik durchgeführt werden können.

operable (bedienbar)
Was Stakeholder unter bedienbar verstehen nach dem arc42 Quality Model:

„(potential) Expectation for operable of Developers:
- *automated test and build*
- *appropriate automation of deployments*
- *appropriate similarity of development and production environments “*

(Starke, #operable, 2022)

„(potential) Expectation for operable of Admin:
- *easy to build and deploy*
- *appropriate monitoring facilities*
- *appropriate procedures for crisis management in place*
- *appropriate management of credentials required“*

(Starke, #operable, 2022)

Bereits zu Beginn der Entwicklung sollen Test- und Deploy-Pipeline eingerichtet werden, welche das Testing und Deployment automatisieren. Dazu gehören auch die Tests, welche parallel zur Entwicklung geschrieben werden sollen.

Ausserdem soll es ein übergreifendes Log-System geben, welches es ermöglicht, technische Fehlerursachen schnell zu finden und zu beheben.

sicher (save)
Was Stakeholder unter sicher verstehen nach dem arc42 Quality Model:

„(potential) Expectation for save of Developer:
- *despite corporate security measures, public sources (like Stack Overflow, GitHub and common search engines) are accessible*
- *automated and tested backup for everything*
- *all important documents and files are version-controlled“*

(Starke, #safe, 2022)

Das arc42 Quality Model hat hierfür keine Definition, was Administratoren unter sicher verstehen.

Wie bereits im vorherigen Abschnitt erwähnt, wird die Automatisierung mittels Test- und Deploy-Pipeline sichergestellt.

geeignet (suitable)
Was Stakeholder unter geeignet verstehen nach dem arc42 Quality Model:

„(potential) Expectation for suitable of Developer:
- *appropriate effort required to understand internals*
- *good code readability*
- *appropriate effort required to locate and fix bugs*
- *appropriate technologies used*
- *appropriate technical documentation“*

(Starke, #suitable, 2022)

„(potential) Expectation for suitable of Admin :
- *easy to perform required administration tasks (like deploy, install, configure etc)“*

(Starke, #suitable, 2022)
Trotz dessen, dass die Middleware nach dem Architektur-vorbild eines modularen Monoliths organisiert ist wird darauf geachtet, dass die Struktur des Codes und die verwendeten Abhängigkeiten grösstenteils dieselben sind.

Auch bei der Strukturierung des Codes und des Programmflusses wird auf eine Einheitlichkeit über die verschiedenen Module des modularen Monoliths geachtet.

10.1.2.3. Konnektoren
Die Zielgruppe der Konnektoren, die die jeweiligen Protokolle und Plattformen mit der Middleware verbinden, sind hauptsächlich technisch interessierte und versierte Anwender. Im Umfeld von arc42 können dies Domänen-Experten oder andere Personen sein.

zuverlässig (reliable)
Was Stakeholder unter zuverlässig verstehen nach dem arc42 Quality Model:

„(potential) Expectation for reliable of others:
- *technical documentation is reliable (current and correct) “ (Starke, #reliable, 2022)*

Dass arc42 Quality Model hat hierfür keine Definition, was Domänen Experten unter zuverlässig verstehen.

Die Anforderungen an die Konnektoren für die Kommunikation mit der Middleware sollen sehr spezifisch und einfach verständlich sein.

brauchbar (usable)
Das arc42 Quality Model hat hierfür keine Definition, was Domänen Experten unter usable verstehen.

Es soll bei Anforderung an die Kommunikation vom Konnektor zur Middleware darauf geachtet werden, dass nur die absolut notwendigen Informationen ausgetauscht werden müssen.

flexibel (flexible)
Das arc42 Quality Model hat hierfür keine Definition was Domänen-Experten und andere unter flexibel verstehen.

Die jeweiligen Konnektoren sollen so wenig Anforderung an die Kommunikation erfüllen wie nur möglich, um so genau auf den Bounded-Context der darunterliegenden Plattform oder des Protokolls angepasst werden zu können.

bedienbar (operable)
Das arc42 Quality Model hat hierfür keine Definition was Domänen Experten und andere unter operable verstehen.

Für die Zielgruppe soll es so einfach wie möglich sein einen Konnektor einzurichten, auch ohne dass sie genau verstehen, wie die Kommunikation im Hintergrund funktioniert.

10.1.2.4. Messaging

Bei der Übermittlung der Nachrichten an den zentralen Server, ist die Zielgruppe das DevOps-Team, welche die Middleware wartet, weiterentwickelt und betreibt. Doch auch der technisch interessierte und versierte Anwender, welcher die Konnektoren betreibt, kommt in Berührung mit der Messaging-Lösung und gehört daher zur Zielgruppe.

zuverlässig (reliable)
Was Stakeholder unter zuverlässig verstehen nach dem arc42 Quality Model:

„(potential) Expectation for reliable of Developers:
- *reliably add new features or functions to the system without unwanted side-effects.*
- *realiably predict the effects of changes to the system“ (Starke, #reliable, 2022)*

„(potential) Expectation for reliable of others:
- *technical documentation is reliable (current and correct)“ (Starke, #reliable, 2022)*

Die Anforderungen an das Messaging sind eine gute Dokumentation und eine einfache Integration in das vorhandene Ecosystem.

brauchbar (usable)
Was Stakeholder unter brauchbar verstehen nach dem arc42 Quality Model:

„(potential) Expectation for usable of Developers:
- *understandable source-code and dependencies*
- *an appropriate technology-stack*
- *no (or at least predictable) side-effects when changing the system“*

(Starke, #usable, 2022)
Das arc42 Quality Model hat hierfür keine Definition was andere unter brauchbar verstehen.

Bei der Auswahl der Messaging-Lösung wird aktiv auf die Unterstützung von JMS geachtet, um die Messaginglösung auch relativ einfach austauschen zu können. Zudem kennen viele Entwickler die Konzepte von JMS und kennen sich so schneller in der Umgebung aus.

flexibel (flexible)
Das arc42 Quality Model hat hierfür keine Definition, was Entwickler und andere unter flexibel verstehen.
operable (bedienbar)
Was Stakeholder unter brauchbar verstehen nach dem arc42 Quality Model:

„(potential) Expectation for operable of Developers:
- *automated test and build*
- *appropriate automation of deployments*
- *appropriate similarity of development and production environments "*

(Starke, #operable, 2022)
Das arc42 Quality Model hat hierfür keine Definition, was andere unter brauchbar verstehen.

effizient (efficient)
Was Stakeholder unter effizient verstehen nach dem arc42 Quality Model:

„(potential) Expectation for reliable of Developers:
- *new features can be implemented quickly*
- *build works quickly*
- *automated tests run quickly"*

(Starke, #efficient, 2022)
Das arc42 Quality Model hat hierfür keine Definition, was andere User unter effizient verstehen.

Die Anforderungen an die Messaging-Lösung sind jene, dass eine grosse Anzahl an Nachrichten mit möglichst kurzer Verzögerung übertragen werden. Zudem soll es bei einem Nachrichten-Stau nicht zu einem Absturz kommen. Das Schreiben und Lesen auf die Messaging-Lösung soll möglichst einfach möglich sein.

10.1.3. Akzeptanzkriterien

Um prüfen zu können, ob die Qualitätsziele auch erfüllt sind, wurden die folgenden Akzeptanz-kriterien definiert.

10.1.3.1. Akzeptanzkriterien der Middleware-API

zuverlässig (reliable)
- Fehlerbehandlung:
  - Die API kommuniziert klar und eindeutig. Wenn etwas schief geht, wird dies mittels standardisierte HTTP-Statuscodes und Fehlermeldungen dem API-Konsumenten mitgeteilt.

brauchbar (usable)
- Entwicklererfahrung:
  - Die API selbst ist in einem branchenüblichen Standard implementiert.
  - Die API-Responses verweisen mittels Hyperlinks auf weitere Endpunkte.

flexibel (flexible)
- Anpassungsfähigkeit:
  - An Endpunkte, die grössere Datenmengen zurückgeben werden in einzelne Seiten aufgeteilt.

operable (bedienbar)

- Entwicklererfahrung:
  - Es wird eine Test-Umgebung zur Verfügung gestellt, um die Endpunkte auszuprobieren. *(nicht Umfang der Bachelor-Arbeit)*

10.1.3.2. Akzeptanzkriterien der Middleware

zuverlässig (reliable)

- Skalierbarkeit:
  - Die Cloud-Applikation muss dynamisch skalierbar sein, um Lastspitzen effektiv zu bewältigen. *(nicht Umfang der Bachelor-Arbeit)*

brauchbar (usable)

- Zugänglichkeit
  - Es kommen branchenübliche Pattern zum Einsatz.

flexibel (flexible)

- Modularität
  - Die Businesslogik ist, basierend auf ihren Funktionen, in einzelne Module gruppiert.

operable (bedienbar)

- Log
  - Ereignisse, die für die Fehlerermittlung relevant sind (Bsp. Exceptions), können geloggt werden.
  - Es können systemübergreifende Ereignisse geloggt werden. So können System-Logs von den Konnektoren an den Server übermittelt werden.

sicher (save)

- API
  - Durch Austauschen einer anderen User-ID können nicht unberechtigt auf fremde Daten zugegriffen werden.

10.1.3.3. Akzeptanzkriterien der Konnektoren

zuverlässig (reliable)

- Leistung:
  - Der Konnektor sollte effizient auf der vorhandenen Hardware laufen.

brauchbar (usable)

- Zugänglichkeit:
  - Ein technisch versierter Benutzer, der mindestens eine technische Lehre erfolgreich abgeschlossen hat, soll konzeptionell verstehen:
    - Was ein Konnektor ist
    - Was dessen Aufgabe ist
    - Wie er mit dem Server kommuniziert

flexibel (flexible)

- Konfigurierbarkeit:
  - Ein Domänen-Experte, der das SmartHome oder IoT-System selbst einrichten kann, sollte das Context-Mapping erstellen können.

operable (bedienbar)

- *Sicherheit:*
  - Die Applikation muss sichere Authentifizierungs- und Autorisierungsmechanismen bieten.

10.1.3.4. Akzeptanzkriterien der Messaging Plattform

zuverlässig (reliable)

- Die Messaging-Lösung verfügt über eine mehrseitige Dokumentation.
- Der Code, um eine Nachricht zu senden oder zu empfangen, kann auf weniger als drei A4 Seiten ausgedruckt werden *(mind. Schriftgrösse 5).*

brauchbar (usable)

- Die übertragenen Nachrichten können ohne spezielle Algorithmen de-serialisiert werden.

Flexible

- Die übertragenen Nachrichten können angepasst werden, um neue Funktionen zu unterstützen, ohne dass Konnektoren von der Kommunikation ausgeschlossen werden.

effizient (efficient)

- Die Messaginglösung kann mit einem Nachrichten-Stau, mehr neue Nachrichten eingehend als ausgehend (doppelte Menge, bei 500 Nachrichten / Minute eingehend) umgehen, ohne abzustürzen (über einen Zeitraum von 30 Sekunden).

11. Risiken und Erkenntnisse

Da sowohl die Hersteller der SmartHome- und IoT-Systeme als auch die User-Interface Entwickler sich im Klaren sein sollten über die vorherrschenden Risiken sowie von den gewonnenen Erkenntnissen im Rahmen dieser Arbeit, richtet sich dieses Kapitel an sie.

Allgemein
Risiken und technische Schulden können eine Herausforderung für die Wartbarkeit und Erweiterbarkeit des Systems darstellen. Daher sollten die Risiken während der gesamten Lebensdauer eines Produktes periodisch überprüft und neu eingeschätzt werden.

11.1. Risiken

Bereits zu Beginn wurde die erste Risikoanalyse durchgeführt. Da wurden auch schon die ersten Risiken identifiziert. Ebenso wurden bei den nächsten Iterationen der Risikoanalysen jeweils weitere Risiken identifiziert.

11.1.1. Identifizierte Risiken der ersten Iteration

Bereits nach dem zweiten Sprint wurden die ersten Risikoanalysen durchgeführt und dabei wurden folgende Risiken identifiziert.

11.1.1.1. Arbeitsumfang von drei Konnektoren

Das Implementieren eines Konnektors impliziert ein grundlegendes Verständnis des Fremd-Systems. Der Umstand, dass es vor einer Generalisierung branchenüblich ist, mindestens dreimal sehr ähnliche Arbeiten durchgeführt zu haben, blieb es nicht bei einem Konnektor, sondern sollten mindestens drei Konnektoren implementiert werden. Diese Tatsache verdreifachte den unbekannten Arbeitsaufwand, was zu einem erheblichen Block an unbekanntem Zeitbedarf darstellt.

11.1.1.2. Software-Änderungen der Geräte

Um die Hersteller davon zu überzeugen, dass sie Ihren Kunden mit der Entwicklung eines Konnektors einen Mehrwert bieten, müssen erste Prototypen davon implementiert werden. Doch kann es dazu kommen, dass sich das Verhalten oder die Endpunkte währenddessen ändern. Dies könnte aufgrund einer Softwareanpassung auf den Fremd-System-Geräten eintreten.

11.1.1.3. API-Anpassungen der Fremd-Systeme

Nicht jedes Fremd-System-Gerät ist selbst direkt erreichbar. Daher sollte auch ein Fremd-System mittels einer Cloud-API angebunden werden. Doch unterliegt die Kontrolle und der Update-Zyklus der API beim Hersteller dieser API.

11.1.1.4. Komplexität

Jedes Fremd-System bringt ein eigenes Verständnis der Funktionalitäten, der Abläufe, der verwendeten Technologien und Bedeutungen der Begriffe mit sich. Aus all diesen Verständnissen muss eine gemeinsame Basis gefunden werden, die am Ende ein Endbenutzer auch noch versteht und als intuitiv empfindet.

11.1.2. Identifizierte Risiken währen der Arbeit

Nach ca. der Hälfte der Zeit bis zur Abgabe der Arbeit wurden die bereits identifizierten Risiken neu beurteilt und weitere Risiken identifiziert, auf welche ich in der ersten Projekthälfte aufmerksam wurde.

11.1.2.1. Update des Risikos «Arbeitsumfang von drei Konnektoren»

Um die Ungewissheit, womit dieses Risiko einhergeht, zu beschränken, wurde bei der Entwicklung darauf geachtet, dass wo immer vorhanden auf Open-Source Projekte zurückgegriffen wurde. Diese sollten die fachliche Kommunikation mit den Fremd-Systemen übernehmen.

So wurde sowohl beim KNX-Konnektor mit dem Calimero-Projekt als auch beim Sonos-Konnektor mittels Sonos API-Client vom GitHub-Benutzer «nightowlengineer» umgesetzt. Beim Shelly-Konnektor musste selbst eine Java-Library zur Kommunikation mit den Shelly-Geräten geschrieben werden.

11.1.2.2. Update des Risikos «Softwareänderungen der Geräte»

Um das Risiko von unerwarteten Änderungen basierend auf Softwareänderungen zu mittigen, wurde darauf geachtet, dass wo immer möglich die automatisierten Updates der Fremd-System-Geräte deaktiviert wurden.

11.1.2.3. Update des Risikos «API-Anpassungen der Fremd-Systeme»

Aus dem Grund, dass die Hersteller die Hoheit über Änderungen an den API's haben, wurde die Verwendung von Could-API's zur Steuerung der Geräte auf ein Fremd-System begrenzt. Dies wurde beim Fremd-System von Sonos umgesetzt.

11.1.2.4. Update des Risikos «Komplexität»

Eines der Ziele der Arbeit ist es, dass der Endbenutzer in das Zentrum rückt und nicht die Technik. Daher wurden bereits zu Beginn der Arbeit Endbenutzer befragt, wie dessen Verständnis der Smart Home und IoT-Komponenten ist.

So wurde zum Beispiel der Begriff «Gerät» im Kontext der Middleware für den im Kontext der Elektronik geltenden Begriff eines «Schalt-Ausganges» gleichgesetzt.

11.1.2.5. Entwickler-Account für das Fremd-System von Sonos

Um die notwendigen Zugangs-Daten zu erlangen, welche für das Steuern der Sonos-Speaker über die Sonos-Music-API notwendig ist, muss ein entsprechender Account erstellt werden. Der Zugang auf diese Seite ist jedoch aufgrund eines technischen Fehlers nicht möglich.

Da ohne diesen Zugang das Benutzen der Sonos-Music-API nicht möglich sein wird, stellt dies ein Risiko für die Aussagekraft des PoC's dar.

11.1.2.6. Zusatzaufwand durch Entscheidung für eine Modulith-Architektur

Basierend auf dem im ADR 9.1.1 - Monolith versus Micro-Services versus Modularer Monolith (Modulith) erfassten Entscheid, die Middleware nach dem Vorbild eines Modulithen zu implementieren, tauchte das Risiko von Verzögerungen durch die verteilte Datenhaltung auf. Denn bei einem Modulithen werden die Daten jedes Modules eigenständig gespeichert und verwaltet. Trotzdem tritt die Middleware nach aussen als eine Applikation auf was dazu führt, dass ein Mechanismus zur Verteilung der relevanten Update-Informationen implementiert werden musste. Dies ging mit einem zusätzlichen nur schwer abschätzbaren Zeitaufwand einher.

11.1.3. Identifizierte Risiken gegen Ende der Arbeit

Ein Sprint vor der Abgabe der Arbeit wurde die letzte Risikoanalyse durchgeführt und dabei wurden folgende Risiken neu eingeschätzt.

11.1.3.1. Update des Risikos «Entwickler-Account für das Fremd-System von Sonos»

Nach mehreren Anläufen und einiger Zeit, wurde der technische Fehler behoben und ich konnte die Zugangsdaten lösen. Dies wandelte dieses Risiko in eine Verzögerung und einen unerwarteten Mehraufwand um.

11.1.3.2. Update des Risikos «Zusatzaufwand durch Entscheidung für eine Modulith-Architektur»

Bei der Implementation des Mechanismus zum Update der Informationen über die verschiedenen Module hinweg wurde darauf geachtet, dass nur die relevanten Informationen synchronisiert werden.

Zudem wurde der Aufwand dem PoC-Status der Arbeit angepasst. Daher wurde auf die Implementierung von ausgeklügelten Konfliktlösungsmechanismen verzichtet.

11.2. Lessons Learned

Dieser Abschnitt befasst sich mit den gewonnenen Erkenntnissen dieser Arbeit und des darin umgesetzten Proof-of-Concept.

11.2.1. Granularität der Module

Die Aufteilung der Funktionalitäten in der Middleware in vier einzelne Funktions-Module war rückblickend wahrscheinlich etwas zu fein-granular. Dies vor allem unter der Berücksichtigung, dass sich die Middleware aktuell noch in einer PoC-Phase befindet. In einem neuen Anlauf würden der Policy-Hub und die Command-Suite sowie der Monitoring-Suite und der Record-Keeper zusammengelegt werden. Dies auch aus dem Grund, dass der Overhead, welcher jedes Modul benötigt im Verhältnis zu dem Funktionsumfang, was im Rahmen einer Bachelor-Einzel-Arbeit umgesetzt werden kann, recht gross ist. Aufgrund der Tatsache, dass dieses Projekt weiterverfolgen wird, ist dies jedoch nicht als Fehlarchitektur, sondern als architektonische Investition in die Zukunft des Projektes, zu werten.

11.2.2. «Rule of three»-Prinzip

Eines der grundlegenden Prinzipien bei der Software-Entwicklung und dem Engineering neuer Systeme besagt, dass erst ab einer Anzahl von drei sehr ähnlichen Umsetzungen einer Arbeit eine Generalisierung sinnvoll ist. Dies da erst nach drei Umsetzungen genügend Erkenntnisse gewonnen werden, um ein gesamtheitliches Bild über die Herausforderungen zu haben.

Basierend auf diesen Regeln wurden drei unterschiedliche Fremd-Systeme evaluiert, welche in Bezug auf die Architektur teilweise Überschneidungen besitzen. Ausserdem war dies auch der Grund, warum zuerst die Konnektoren implementiert und erst danach die Middleware designt wurde.

Rückblickend hat sich dieser Ansatz in Bezug auf die gewonnenen Erkenntnisse über die Herausforderungen sehr bewährt. Bezogen auf das zeitliche Budget, welches bei dieser Einzelarbeit 360 Stunden betrug, war die Implementation von drei Konnektoren eine Entscheidung die zur Folge hatte, dass in anderen Bereichen Abstriche gemacht werden musste.

11.2.3. Geschlossene Fremd-Systeme

Während der Evaluation der Fremd-Systeme wurde bald klar, dass nicht jeder Hersteller ein Interesse daran besitz die Steuerung der Geräte und die Herausgabe an Sensor-Informationen zu unterstützen. Dies betraf immer die vom Kunden bereits gekauften Systeme und seine Datenpunkte.

Dies lässt die Vermutung zu, dass sich die Hersteller bei der User-Experience ihrer Kunden nicht in ein Risiko begeben wollen, um keine Enttäuschungen zu erleiden. Dies ist aus diesem Blickwinkel verständlich, da es speziell bei SmartHome-Systemen in den vergangenen Jahren immer wieder zu Fehlern und ungewolltem Verhalten gekommen ist, das darauf zurückzuführen ist, dass der Endbenutzer zu wenig im Mittelpunkt der Entwicklung gestanden hat.

11.3. Rückblick über PoC

Es darf ein positives Fazit aus dem PoC gezogen werden. Nicht nur dass der PoC gezeigt hat, dass es möglich ist dem Endbenutzer ein einheitliches User-Interface zur Verfügung zu stellen, sondern auch dass die Hoheit der einzelnen Software-Komponenten verteilt werden kann. Dies sowohl bei der Implementation durch die Herstellung von Konnektoren durch den Fremd-System-Hersteller und die Middleware als mögliches Open-Source-Projekt, als auch während dem Betrieb, indem die Konnektoren lokal im Netzwerk des Endbenutzers und die Middleware in der Cloud laufen.

Die Überzeugung, dass eine solche Middleware Erfolg haben kann, wurde nun noch weiter bekräftigt. Wichtig hierbei ist jedoch, dass die Sicht des Kunden und nicht die der technischen Umsetzung in den Fokus rückt. Aufgrund der Tatsache, dass die Smart Home- und IoT-Industrie ein sehr stark umkämpfter Markt ist, sehe ich die Zukunft einer solchen Middleware nicht in einer Unternehmung, sondern in einem Open-Source-Projekt.

Aufgrund des begrenzten Zeitbudgets von ca. 360 Arbeitsstunden mussten sowohl beim Testing als auch bei den nicht-funktionalen Anforderungen Abstriche gemacht werden. Da ich dieses Projekt jedoch weiterhin betreue, kann ich diese Anforderungen ausserhalb der Arbeit umsetzen.

11.4. Ausblick

Die nächsten Konnektoren (z.B. Victron energy) sind bereits in Planung. Auch weitere Funktionalitäten auf der Middleware (z.B. wetterbasierte Regeln) selbst sind in Planung und werden in den nächsten Sprints umgesetzt werden.

11.4.1. Weitere Konnektoren

Um weitere Fremd-Systeme anbinden zu können, werden weitere Konnektoren benötigt. Daher wurden bereits weitere potenzielle Kandidaten evaluiert.

11.4.1.1. Victron energy

Victron energy ist ein Hersteller von elektronischen und elektrischen Komponenten. Meist werden deren Komponenten (Wechselrichter für PV-Anlagen oder Diesel-Generatoren, Batterie-Ladegeräte und Monitore etc.), auf Fahrzeugen oder an abgelegenen Orten eingesetzt. Um die Überwachung der Geräte sicherstellen zu können stellt der Hersteller seinen Kunden die VRM-API zur Verfügung. Darüber können nahezu alle Daten des Systems ausgelesen und angepasst werden.

11.4.2. Weitere Funktionalitäten auf der Middleware

Nebst der zuvor ausgeführten Erweiterung des Konnektoren-Bestandes, sollen auch die nachfolgenden Funktionalitäten in naher Zukunft implementiert werden.

11.4.2.1. Wetterbasierte Regeln für den Policy-Hub

Um auch wetterbasierte Regeln für die Automatisierung hinzuziehen zu können, soll eine Wetter-API angebunden werden, basierend auf deren Informationen (des zukünftigen oder des aktuellen Wetters) reagiert werden soll.

11.4.2.2. Sonnenstand-basierte Regeln für den Policy-Hub

Für die Steuerung von Belüftungen oder drehbaren Solaranlagen ist der Stand und der Winkel der Sonne von Bedeutung. Daher sollen auch basierend auf diesen Informationen Aktionen im Policy-Hub erfasst werden können.

11.4.2.3. Exportfunktion des Record-Keepers

Da bei einer grösseren Gesamtanlage es dazu kommen kann, dass die Records im Record-Keeper, welcher die Aktionen dauerhaft persistiert, auch ausserhalb der Middleware ausgewertet werden möchten, sollen diese in verschiedenen Dateiformaten wie zum Beispiel CSV, JSON oder XML exportiert werden können.

11.4.2.4. Verweildauer der Events auf dem Record-Keeper

In den meisten Fällen sind die Events, welche innerhalb eines Smart Home- oder IoT-Systems erfolgen, nach Ablauf von 14 Tagen nicht mehr von Relevanz. Daher ist eine Einstellung in Planung, bei welcher die Verweildauer der Daten im Record-Keeper individualisiert werden kann. Dies könnte auch Bestandteil eines möglichen Abo-Modells sein.

11.4.2.5. Sensoren für die Monitoring-Suite

Zusätzlich zu den Zustands-Anzeigen sollen auch Temperatur-Sensoren über die Monitoring-Suite-API abgefragt werden können.

12. Literaturverzeichnis

Hohpe, G., & Woolf , B. (10. 10 2003). *Enterprise Integration Patterns*. Abgerufen am 08. 06 2024 von https://www.enterpriseintegrationpatterns.com/patterns/messaging/EnvelopeWrapper.html

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1997). *Design patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley.

Hohpe, G., & Woolf , B. (10. 10 2003). *Enterprise Integration Patterns*. Abgerufen am 09. 06 2024 von https://www.enterpriseintegrationpatterns.com/patterns/messaging/PublishSubscribeChannel.html

Starke, G. (2022). *arc42 Quality Model*, generiert am 28. April - 13:34. Abgerufen am 09. 06 2024 von https://quality.arc42.org/tag-reliable/

Starke, G. (2022). *arc42 Quality Model*, generiert am 28.April - 13:34. Abgerufen am 09. 06 2024 von https://quality.arc42.org/

Starke, G. (2022). *arc42 Quality Model*, generiert am 28. April - 13:34. Abgerufen am 09. 06 2024 von https://quality.arc42.org/tag-usable/

Starke, G. (2022). *arc42 Quality Model*, generiert am 28. April 13:34. Abgerufen am 09. 06 2424 von https://quality.arc42.org/tag-operable/

Starke, G. (2022). *arc42 Quality Model*, generiert am 28. April 13:34. Abgerufen am 09. 06 2024 von https://quality.arc42.org/tag-suitable/

Starke, G. (2022). *arc42 Quality Model*, generiert am 28. April 13:34. Abgerufen am 09. 06 2024 von https://quality.arc42.org/tag-efficient/

Starke, G. (2022). *arc42 Quality Model*, generiert 28. April - 13:34. Abgerufen am 09. 06 2024 von https://quality.arc42.org/tag-safe/

Zimmermann, O., Stocker, M., Lübke, D., Zdun, U., & Pautasso, C. (2022). *Patterns for API Design*. Abgerufen am 08. 06 2024 von Pattern: Information Holder Resource: https://microservice-api-patterns.org/patterns/responsibility/endpointRoles/InformationHolderResource

Zimmermann, O., Stocker, M., Lübke, D., Zdun, U., & Pautasso, C. (2022). *Patterns for API Design*. Abgerufen am 08. 06 2024 von Pattern: Atomic Parameter: https://microservice-api-patterns.org/patterns/structure/representationElements/AtomicParameter

Zimmermann, O., Stocker, M., Lübke, D., Zdun, U., & Pautasso, C. (2022). *Patterns for API Design*. Abgerufen am 08. 06 2024 von Pattern: Processing resource: https://www.microservice-api-patterns.org/patterns/responsibility/endpointRoles/ProcessingResource

Zimmermann, O., Stocker, M., Lübke, D., Zdun, U., & Pautasso, C. (2022). *Patterns for API Design*. Abgerufen am 08. 06 2024 von Pattern: Embedded Entity: https://microservice-api-patterns.org/patterns/quality/referenceManagement/EmbeddedEntity

Zimmermann, O., Stocker, M., Lübke, D., Zdun, U., & Pautasso, C. (2022). *Patterns for API Design*. Abgerufen am 08. 06 2024 von Pattern: Linked Information Holder: https://www.microservice-api-patterns.org/patterns/quality/referenceManagement/LinkedInformationHolder

Zimmermann, O., Stocker, M., Lübke, D., Zdun, U., & Pautasso, C. (2022). *Patterns for API Design*. Abgerufen am 06. 10 2024 von Pattern: Pagination: https://microservice-api-patterns.org/patterns/quality/dataTransferParsimony/Pagination

13. Glossar

Das Glossar richtet sich an die Zielgruppe der Endbenutzer und verfügt über den Charakter als Nachschlagewerk für verwendete und branchenübliche Fachbegriffe und Abkürzungen.

13.1. Abkürzungsverzeichnis

A		
ACID	-	Atomicity, Consistency, Isolation, Durability
ACL	-	Anticorruption Layer
ADR	-	Architecture decision record
AES	-	Advanced Encryption Standard
API	-	Application Programming Interface
B		
BC	-	Bounded-Context
D		
DDD	-	Domain-Driven-Design
H		
HATEOAS	-	Hypermedia as the Engine of Application State
HTTP	-	Hypertext Transfer Protocol
I		
IANA	-	Internet Assigned Numbers Authority
IoT	-	Internet of Things
J		
JPA	-	Jakarta Persistence API
JSON	-	JavaScript Object Notation
O		
OHS	-	Open Host Service
P		
POC	-	Proof-of-Concept
R		
RPC	-	Remote Procedere Call
RSA	-	Rivest, Shamir, Adleman-Verfahren
S		
SQL	-	Structured Query Language
SSL	-	Secure Sockets Layer
SSOT	-	Single source of truth
U		
URL	-	Uniform Resource Locator
X		
XML	-	Extensible Markup Language
Y		
YAGNI	-	YAGNI-Prinzip

13.2. Fachbegriffe

A

13.2.1.1. ActiveMQ Artemis

ActiveMQ ist eine Messaging-Plattform, welche unter der Aufsicht der Apache Software Fundation entwickelt wird und verschiedene Protokolle unterstützt. Die neueste Weiterentwicklung von ActiveMQ ist die ActiveMQ Artemis, welche noch leistungsfähiger ist und bei dieser Arbeit eingesetzt wurde.

13.2.1.2. Azure Container Instance

Die Container Instance ist eine Dienstleistung des Microsoft Cloud-Portals Azure, welche sich jedermann öffentlich durch einen Container-Registry zugänglichen Docker-Container kopieren und für den Kunden betreiben kann.

13.2.1.3. Advanced Encryption Standard – AES

AES ist ein symmetrischer Verschlüsselungsalgorithmus, der von den USA als Standard für die Verschlüsselung sensibler Daten angenommen wurde. AES unterstützt Schlüsselgrössen von bis zu 256 Bit, was ihn sowohl sicher als auch effizient für verschiedene Anwendungen macht.

13.2.1.4. Atomicity, Consistency, Isolation, Durability – ACID

Das Akronym-ACID steht für:

- **A**tomicity - Atomarität
  beschreibt, dass eine Datenbank-Transaktion vollständig oder gar nicht durchgeführt wurde und bei nicht vollständiger Durchführung die Änderungen wieder rückgängig gemacht werden.

- **C**onsistency - Konsistenz
  besagt, dass die Datenbank nach der Transaktion in einem gültigen Zustand ist.

- **I**solation - Isolation
  stellt sicher, dass die Transaktionen untereinander keinen Einfluss aufeinander ausüben.

- **D**urability - Dauerhaftigkeit
  steht dafür, dass nach Abschluss der Transaktion der neue Bestand der Daten in der Datenbank dauerhaft enthalten ist.

13.2.1.5. Architecture decision record – ADR

Ein «Architecture decision record» dient zur Dokumentation einer Entscheidung, welche die Architektur oder einen Einfluss darauf hat. Dazu wird in einfacher Form formell festgehalten, auf welchem Kontext welche Entscheidung unter den entsprechenden Aspekten, wie Vor- und Nachteile sowie deren Folgen, getroffen wurde. Dies fördert sowohl die Nachvollziehbarkeit als auch den Knowhow-Transfer.

13.2.1.6. Anticorruption Layer – ACL

Ein Anticorruption Layer ist ein Design-Ansatz, das als Isolationsschicht zwischen zwei Bounded-Contexts aus dem strategischem Domain-Driven-Design.

13.2.1.7. Application-Event-Publisher

Ein Application-Event-Publisher ist eine Komponente, die in der Spring-Dependency «Modulith» enthalten ist und eine applikations-interne Messaging-Plattform nach dem Public-Subscribe-Pattern zur Verfügung stellt.

13.2.1.8. Application Programming Interface

Eine API ist eine Schnittstelle über welche aussenstehenden Personen oder Applikationen mit der Applikation, welche die API zur Verfügung stellt, interagieren kann. Dies geschieht meist nach dem Frage-Antwort Prinzip.

B

13.2.1.9. Bounded-Context - BC

Der Bounded-Context ist ein Konzept, das aus dem Domain-Driven-Design kommt und ein Bereich abgrenzt, der auch Domäne genannt wird, indem eine Ansammlung von Begriffen eine bestimmte Bedeutung besitzen. Diese Ansammlung von Begriffen wird im Kontext des Domain-Driven-Designs auch Ubiquitous-language genannt.

C

13.2.1.10. Conformist – CF

Ein Conformist ist ein Konzept aus dem Domain-Driven-Design, bei dem der Conformist den Bounded-Context des Systems übernimmt, das sich auf der anderen Seite der API befindet.

D

13.2.1.11. (Maven-)Dependency

Eine Dependency ist eine Applikations-Abhängigkeit, welche von der Applikation benötigt wird, um wie programmiert zu funktionieren. Hierzu wird der Code der Dependency in die Applikation hineingeladen und beim Bauen der Applikation mit ausgeliefert.

13.2.1.12. Deployment

Das Deployment bezeichnet den Prozess des ausliefern einer Applikation in eine Produktionsumgebung.

13.2.1.13. (Docker-)Container

Ein Container ist eine eigenständig ausführbare Softwareeinheit, welche in sich geschlossen lauffähig ist. Er enthält alle dafür notwendigen Komponenten wie die Software selbst, die dazu nötigen Dependencies sowie die Laufzeitumgebung.

13.2.1.14. Domain-Driven-Design - DDD

DDD ist ein Ansatz zur Softwareentwicklung, bei dem der Schwerpunkt auf der Modellierung von Software basierend auf den Kernkonzepten und Logiken eines spezifischen Geschäftsfeldes (der Domain) liegt.

13.2.1.15. Container-Image

Ein Container-Image ist eine Abbildung eines fertigen Containers, welche eine Bauanleitung für andere Container-Laufzeitumgebungen ist, um mit demselben Verhalten zu erstellen.

13.2.1.16. Container-Registry

Eine Container-Registry ist im Zusammenhang mit Containern ein Ablageort für meist öffentlich zugängliche Container-Images.

G

13.2.1.17. Geospatial Data

Unter Geospatial Data (Geodaten) versteht man Informationen über geografische Orte und deren Eigenschaften, die häufig in Form von Koordinaten angegeben werden.

H

13.2.1.18. Hypermedia as the Engine of Application State -HAETOS

HATEOAS ist ein Architekturprinzip im REST-Stil, bei dem der Client die Anwendung über Hypermedia-Navigation steuern kann und den Anwendungszustand durch eingebettete Links und Aktionen ändert.

13.2.1.19. Hypertext Transfer Protocol – HTTP

HTTP ist ein Protokoll zur Übertragung von Hypertext-Dokumenten im World Wide Web und bildet die Grundlage für die Kommunikation im Internet.

13.2.1.20. HTTP-Methoden

HTTP-Methoden wie GET, POST, PUT und DELETE definieren die verschiedenen Aktionen, die auf Ressourcen des Webservers ausgeführt werden können.

I

13.2.1.21. Internet Assigned Numbers Authority

Die IANA ist eine Organisation, die globale IP-Adressen, DNS-Root-Zonen und andere Internet-Protokoll-Ressourcen verwaltet.

13.2.1.22. Internet of Things

Das Internet der Dinge (IoT) vernetzt physische Objekte, die Daten sammeln und austauschen, miteinander und mit dem Internet.

J

13.2.1.23. Jakarta Persistence – JPA

JPA ist eine Java-API, die die Verwaltung und Persistenz von Daten in relationalen Datenbanken durch eine standardisierte Schnittstelle vereinfacht.

13.2.1.24. JavaScript Object Notation – JSON

JSON ist ein leichtgewichtiges Datenformat zur Darstellung strukturierter Daten in lesbarer Textform, das häufig für den Datenaustausch zwischen Client und Server verwendet wird.

K

13.2.1.25. Key-Value-Store

Ein Key-Value-Store ist eine Art NoSQL-Datenbank, die Daten als Schlüssel-Wert-Paare speichert, was schnelle Lese- und Schreiboperationen ermöglicht.

L

13.2.1.26. Latenz

Latenz beschreibt die Verzögerung zwischen dem Senden einer Anfrage und dem Empfang der Antwort, ein wichtiger Faktor für die Leistung von Netzwerksystemen.

M

13.2.1.27. Man-in-the-Middle Angriff

Ein Man-in-the-Middle-Angriff ist eine Sicherheitsbedrohung, bei der ein Angreifer die Kommunikation zwischen zwei Parteien abfängt und möglicherweise manipuliert.

13.2.1.28. Map-Datenstruktur

Eine Map-Datenstruktur speichert Paare von Schlüsseln und Werten, wobei jeder Schlüssel eindeutig ist und auf genau einen Wert zeigt.

13.2.1.29. Maven

Maven ist ein Build-Management-Tool für Java-Projekte, das Projekt-Setup, Abhängigkeiten und Build-Prozesse automatisiert.

13.2.1.30. Microservice-Architektur

Die Microservice-Architektur ist ein Ansatz zur Softwareentwicklung, bei dem Anwendungen aus einer Sammlung kleiner, unabhängiger Dienste bestehen, von denen jeder eine bestimmte Funktion ausführt.

13.2.1.31. Monolithische-Architektur

Die monolithische Architektur ist ein traditioneller Ansatz zur Softwareentwicklung, bei dem eine Anwendung als ein einziges, unteilbares Stück Software entwickelt, ausgeliefert und betrieben wird.

O

13.2.1.32. Observer-Pattern

Das Observer-Pattern ist ein Entwurfsmuster, bei dem ein Subjekt eine Liste von Beobachtern verwaltet und Benachrichtigungen an alle Beobachter sendet, wenn sich der Zustand des Subjekts ändert.

13.2.1.33. Open-host Service

Ein Open-Host-Service ist ein Dienst, der für die Nutzung durch verschiedene Clients offen ist und eine standardisierte Schnittstelle (meist in Form einer API) für die Interaktion bietet.

P

13.2.1.34. Proof-of-Concept – PoC

Ein Proof of Concept ist ein Prototyp oder eine Demonstration die aufzeigt, dass eine Idee, Technologie oder Methode praktisch umsetzbar und nützlich ist.

13.2.1.35. Public-Subscribe-Pattern

Das Publish-Subscribe-Pattern ist ein Nachrichtenmuster, bei dem Nachrichten von Publishern an einen oder mehrere Subscriber weitergeleitet werden, ohne dass diese direkt miteinander kommunizieren.

13.2.1.36. Public Language – PL

Eine Public Language ist eine gemeinsame, domänenspezifische Sprache, die von allen Stakeholdern einer Domäne verwendet wird, um Konzepte und Anforderungen zu kommunizieren.

Q

13.2.1.37. Queue-Datenstruktur

Eine Queue-Datenstruktur ist eine Sammlung von Elementen, die nach dem FIFO-Prinzip (First In, First Out) organisiert ist, wobei das zuerst eingefügte Element auch zuerst wieder entfernt wird.

R

13.2.1.38. Relationale Datenbank

Eine relationale Datenbank speichert Daten in Tabellen und ermöglicht komplexe Abfragen und Beziehungen zwischen den Daten mittels SQL.

13.2.1.39. Remote Procedere Call – RPC

RPC ist ein Protokoll, das es einem Programm ermöglicht, Funktionen auf einem anderen Gerät so auszuführen, als wären es lokale Aufrufe.

13.2.1.40. RESTful HTTP Web API

Eine RESTful HTTP Web API ist eine Schnittstelle für Web Services, die HTTP-Methoden verwendet, um Ressourcen zu manipulieren und eine zustandslose Kommunikation zwischen Client und Server zu ermöglichen.

13.2.1.41. Rivest, Shamir, Adleman-Verfahren – RSA

RSA ist ein kryptographisches Verfahren zur sicheren Übertragung von Daten, das auf asymmetrischer Verschlüsselung und digitalen Signaturen basiert.

S

13.2.1.42. Secure Sockets Layer – SSL

SSL ist ein Sicherheitsprotokoll zur Verschlüsselung von Datenübertragungen im Internet, das die Vertraulichkeit und Integrität der Kommunikation sicherstellt.

13.2.1.43. Set-Datenstruktur

Eine Set-Datenstruktur ist eine Sammlung von eindeutigen Elementen, für die keine doppelten Werte zulässig sind.

13.2.1.44. Separation of Concerns

Separation of Concerns ist ein Entwurfsprinzip, das die Aufteilung einer Software in verschiedene Bereiche vorschlägt, von denen jeder eine bestimmte Aufgabe oder Verantwortung hat, um die Wartbarkeit zu verbessern.

13.2.1.45. Shared Kernel

Ein Shared Kernel ist ein Teil einer Software, der von mehreren Teilsystemen gemeinsam genutzt wird und die Integrität und Konsistenz dieser Teilsysteme sicherstellt.

13.2.1.46. Single Point of Failure – SpoF

Ein Single Point of Failure ist eine Komponente in einem System, deren Ausfall zum Ausfall des gesamten Systems führt.

13.2.1.47. Single source of truth

Single Source of Truth ist ein Konzept, bei dem alle Daten in einem System aus einer einzigen, zuverlässigen Datenquelle stammen, um Konsistenz und Genauigkeit zu gewährleisten.

13.2.1.48. Spring-Modulith

Spring-Modulith ist ein Framework für die Entwicklung modularisierter Spring-Anwendungen, das die Strukturierung und Verwaltung von Modulen innerhalb einer Anwendung erleichtert.

13.2.1.49. Stored Procedures

Stored Procedures sind vordefinierte SQL-Befehlssequenzen, die in einer Datenbank gespeichert und als Einheit ausgeführt werden, um komplexe Operationen zu vereinfachen und die Performance zu verbessern.

13.2.1.50. Structured Query Language – SQL

SQL ist eine standardisierte Programmiersprache zur Verwaltung und Abfrage von Daten in relationalen Datenbanken.

U

13.2.1.51. Ubiquitous language

Eine Ubiquitous Language ist eine gemeinsame Sprache, die von allen Teammitgliedern in einem Softwareprojekt verwendet wird, um Domänenkonzepte und Anforderungen klar und verständlich zu kommunizieren.

13.2.1.52. Uniform Resource Locator – URL

Eine URL ist eine standardisierte Adressierungsmethode, die den Ort einer Ressource im Internet angibt und den Zugriff darauf ermöglicht.

V

13.2.1.53. Virtuelle Threads

Virtuelle Threads sind leichtgewichtige Threads, die von der Laufzeitumgebung verwaltet werden und eine effizientere Nutzung von Systemressourcen für nebenläufige Programmierung ermöglichen.

W

13.2.1.54. Web-Socket

Web-Sockets sind ein Kommunikationsprotokoll, das eine bidirektionale, persistente Verbindung zwischen Client und Server ermöglicht und ideal für Echtzeitanwendungen ist.

X

13.2.1.55. Extensible Markup Language – XML

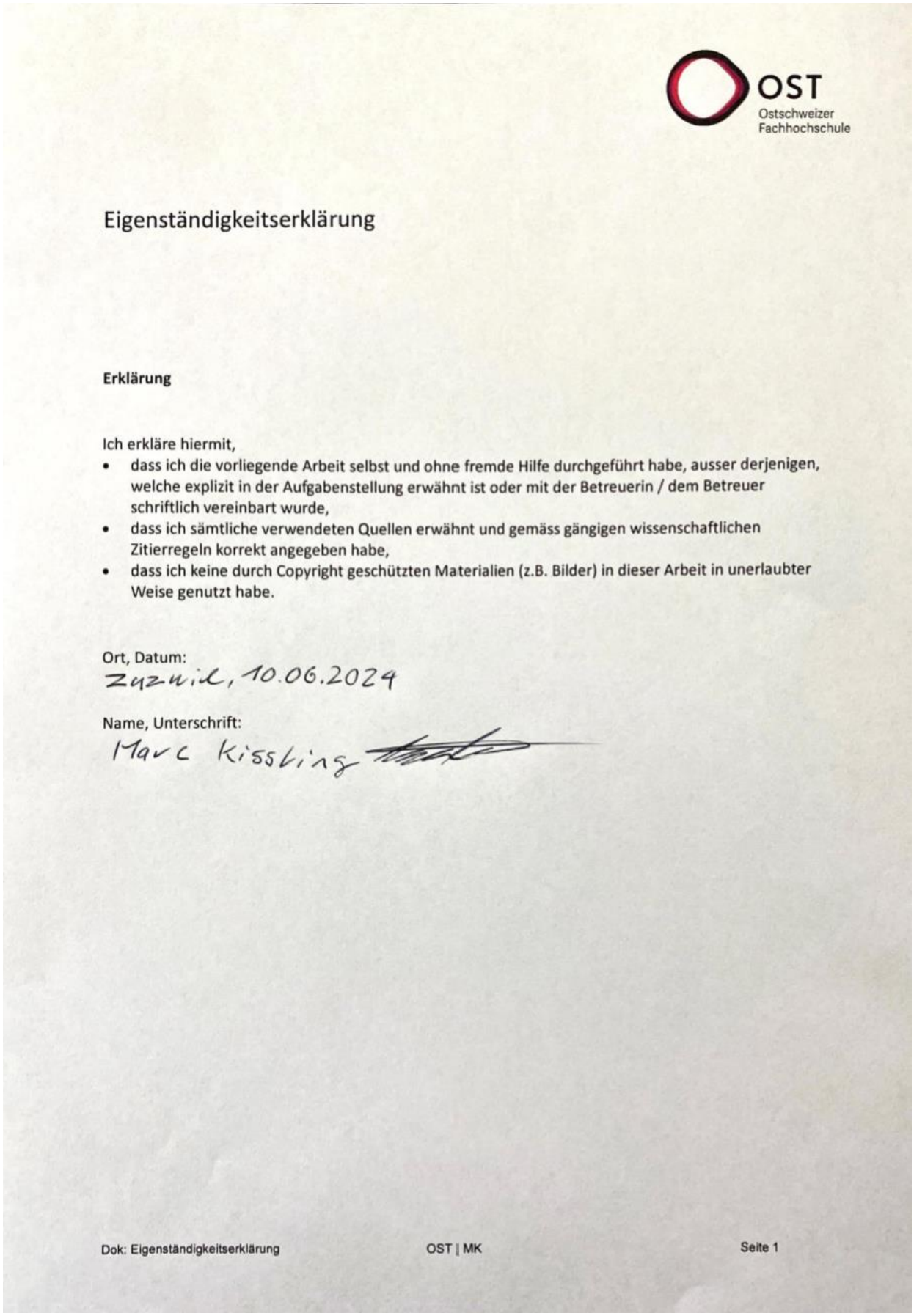
XML ist eine Auszeichnungssprache zur Darstellung hierarchischer Datenstrukturen in einem menschen- und maschinenlesbaren Format, die häufig für den Datenaustausch verwendet wird.

Y

13.2.1.56. YAGNI-Prinzip

Das YAGNI-Prinzip (You Aren't Gonna Need It) besagt, dass Funktionen nur dann implementiert werden sollen, wenn sie tatsächlich benötigt werden, um eine Überentwicklung zu vermeiden.

15. Eigenständigkeitserklärung



OST Ostschweizer Fachhochschule

Eigenständigkeitserklärung

Erklärung

Ich erkläre hiermit,

- dass ich die vorliegende Arbeit selbst und ohne fremde Hilfe durchgeführt habe, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit der Betreuerin / dem Betreuer schriftlich vereinbart wurde,
- dass ich sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben habe,
- dass ich keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt habe.

Ort, Datum:
Zuzwil, 10.06.2024

Name, Unterschrift:
Marc Kissling

Dok: Eigenständigkeitserklärung OST | MK Seite 1